Adobe

Adobe FrameMaker

Structured Application Developer's Reference

March 2026

Contents

Before You Begin 5

Chapter 1 *Structure Application Definition Reference* 9

Contents of an application definition file 9
Define an application 9
Providing default information 11
Specifying the character encoding for SGML files . 11
Specifying conditional text output 13
Specifying a DOCTYPE element 13
Specifying a DTD 14
Specifying entities 14
Specifying entities through an entity catalog . 15
Specifying the location of individual entities . 16
Specifying names for external entity files. . . 17
Specifying public identifiers 18
Specifying a search path for external entity files 19
Specifying external cross reference behavior . . 20
Change file extension to .XML 20
Try alternative extensions 21
Specifying filename extensions 21
Enabling namespaces 22
Specifying a read/write rules document 22
Specifying a search path for included files in rules documents. 22
How FrameMaker searches for rules files. . . 23
Specifying a Schema for XML 23
Specifying an SGML declaration 24
Managing CSS import/export and XSL transformation 24
How the Stylesheets element affects CSS generation. 24
How the Stylesheets element affects CSS import 25
How the Stylesheets element affects XSL transformation 25
Specifying a FrameMaker template 27
Specifying a structure API client 27
Specifying the character encoding for XML files . 27
Display encoding 28
Encoding of CSS files 29
Exporting XML 30
Limiting the length of a log file 30
Mapping graphic notations to file types 31
Defining a Form view for the structured application 31
Specifying MathML options 32

Chapter 2 *Read/Write Rules Summary* 33

All Elements 33
Attributes 34
Books 35
Cross-references 35
Entities 36
Equations 36
Footnotes 37
Graphics 37

Markers 38
Processing instructions 39
Markup documents 39
Tables 40
Text 41
Text insets 41
Variables 41

Chapter 3 *Read/Write Rules Reference* 43

anchored frame 43
attribute 46
character map 49
convert referenced graphics 51
do not include dtd 52
do not include sgml declaration 53
do not output book processing instructions . . . 53
drop 53
drop content 55
element 56
end vertical straddle 59
entity 61
entity name is 63
equation 65
export dpi is 66
export to file 69
external data entity reference 72
external dtd 73
facet 74
fm attribute 76
fm element 77
fm marker 79
fm property 80
fm variable 92
fm version 93
generate book 93
implied value is 97
include dtd 98
include sgml declaration 100
insert table part element 101
is fm attribute 104
is fm char 107
is fm cross-reference element 109
is fm element 110
is fm equation element 111
is fm footnote element 113
is fm graphic element 114
is fm marker element 115
is fm property 116
is fm property value 126
is fm reference element 128
is fm rubi element 130
is fm rubi group element 131
is fm system variable element 132
is fm table element 133
is fm table part element 135
is fm text inset 136
is fm value 139
is fm variable 140
is processing instruction 141
line break 142
marker text is 143

notation is 145
output book processing instructions 147
preserve fm element definition 148
preserve line breaks 149
processing instruction 151
proportional width resolution is 152
put element 153
reader 153
reformat as plain text 154
reformat using target document catalogs . . . 155
retain source document formatting 156
specify size in 157
start new row 159
start vertical straddle 160
table ruling style is 161
unwrap 162
use processing instructions 163
use proportional widths 164
value 165
value is 166
write structured document 166
write structured document instance only . . . 167
writer 168

Chapter 4 *Conversion Tables for Adding Structure to Documents* 171

How a conversion table works 171
Setting up a conversion table 172
Generating an initial conversion table 173
Setting up a conversion table from scratch . . 174
Updating a conversion table 174
Adding or modifying rules in a conversion table . 175
About tags in a conversion table 175
Specifying the root element for a structured document 176
Identifying a document object to wrap . . . 177
Identifying an element to wrap 178
Identifying a sequence to wrap 179
Providing an attribute for an element 181
Using a qualifier with an element 181
Handling special cases 183
Promoting an anchored object 183
Flagging format overrides 184
Wrapping untagged formatted text 184
Nesting object elements 185
Building table structure from paragraph format tags 186
Testing and correcting a conversion table . . . 186

Chapter 5 *CSS to EDD Mapping* 189

CSS 3 to EDD Mapping 189
Basic Properties 189
CSS Selectors 201
Other examples 202
CSS 2 to EDD Mapping 206
CSS Font Properties 206
CSS text properties 208
CSS color and backgrounds properties . . . 210
CSS Formatting Model 211
CSS Pagination Properties 214
CSS generated content, automatic numbering, and lists 215
CSS Tables 217
CSS Selectors 218

Chapter 6 *XML Schema to DTD Mapping* 221

Schema location 221
Namespace and Schema location attributes . . .222
Simple type mapping 222
Attributes of simple type elements 223
Complex type mapping 224
Group 224
Sequence 224
Choice 225
All 226
Named complex types 226
Named attribute groups 227
Abstract elements 228
Mixed content models 229
Supported Schema features 230
Defaults. 230
Any 230
Extension and restriction of complex types . .231
Include, import, and redefine 231
Unsupported Schema features 233

Chapter 7 *The CALS/OASIS Table Model* 235

FrameMaker properties that DO NOT have corresponding CALS attributes 235
Element and attribute definition list declarations .236
Element structure 238
Attribute structure 239
Inheriting attribute values 239
Orient attribute 239
Straddling attributes 239

Chapter 8 *Read/Write Rules for the CALS/OASIS Table Model* 241

Chapter 9 *SGML Declaration* 245

Text of the default SGML declaration 245
SGML concrete syntax variants 247
Unsupported optional SGML features. 248

Chapter 10 *ISO Public Entities* 249

What you need to use ISO public entities 250
Entity declaration files 251
Entity read/write rules files 251
What happens with the declarations and rules . .254

Chapter 11 *Character Set Mapping* 257

Glossary 265

Index 273

Legal notices 279

Before You Begin

This developer reference and its associated developer guide are for anybody who develops structured FrameMaker® templates and XML or SGML applications. They are not written for end users who author structured documents that use such templates and applications.

XML and SGML

FrameMaker can read and write XML (Extensible Markup Language) and SGML (Standard Generalized Markup Language) documents. XML and SGML are both document markup languages, and FrameMaker handles these markup languages in similar ways. However, there are differences between the two, and this manual covers these differences whenever necessary.

When discussing the similarities between them, this manual refers to XML and SGML data as *markup data* or *markup documents*. Otherwise, the manual refers to XML and SGML specifically to draw attention to the differences between these markup languages. The majority of new structured documentation projects are XML based, therefore XML now takes precedence over SGML where necessary.

Developing structured FrameMaker templates

End users of FrameMaker can read, edit, format, and write structured documents—the structure is represented by a hierarchical tree of elements. Each structured document is based on a template that contains a catalog of element definitions. Each element definition can describe the valid contexts for an element instance, and the formatting of element instances in various contexts.

To support these end users, you create the catalog and accompanying structured template.

Developing XML and SGML applications

When FrameMaker reads markup data, it displays that data as a formatted, structured document. When the software saves a structured FrameMaker document, the software can write the document as XML or SGML.

For the end user, this process of translation between FrameMaker documents and markup data is transparent and automatic. However, for most XML or SGML document types the translation requires an XML or SGML application to manage the translation. You develop this application to correspond with specific document types. When your end user opens a markup document with a matching document type, FrameMaker invokes the appropriate structure application. If there is no

match for a document type, the user can choose the application to use, or open the markup document with no structure application.

A structure application primarily consists of:

- A structured template
- DTD or schema
- Read/Write rules (described in this manual)
- XSLT style sheets for pre and post process transformations (if necessary)
- An XML and SGML API client (if necessary) developed with the Frame® Developer's Kit (FDK).

Prerequisites

The following topics, which are outside the scope of this manual, are important for you to understand before you try to create a structured template or structure application:

- Structured document authoring in FrameMaker
- XML or SGML concepts and syntax, including how to work with a *document type definition*
- FrameMaker end-user concepts and command syntax
- FrameMaker template design.

In creating some XML or SGML applications, you may also need to understand the following:

- XSLT 1.0
- C programming
- FDK API usage.

If your application requires only the special rules described in this manual to modify the default behavior of FrameMaker, you do not need programming skills. However, if you need to create an XML and SGML API client to modify this behavior further, you need to program the client in C, using the FDK. This manual does not discuss the creation of XML and SGML API clients. For this information, see the *Structure Import/Export API Programmer's Guide*.

Using FrameMaker documentation

FrameMaker comes with a complete set of end-user and developer documentation with which you should be familiar. You can access the FrameMaker guides from the FrameMaker help and support page, http://www.adobe.com/support/framemaker/.

If you use the Frame Developer's Kit in creating your structure application, you'll also need to be familiar with the FDK documentation set.

Using this manual

This manual provides detailed reference information for application rules and properties. It can be used in conjunction with the Structure Application Developer Guide. It does not currently include EDD reference information. All EDD descriptive and reference information will be found in the Developer Guide.

Typographical conventions

`Monospaced font`	Literal values and code, such as XML, SGML, read/write rules, filenames, and pathnames.
Italics	Variables or placeholders in code. For example, in `name="`*`myName`*`"`, the text *`myName`* represents a value you are expected to supply. Also indicates the first occurrence of a new term.
Blue text	A hyperlink you can click to go to a related section in this book or to a URL in your web browser.
Sans-serif bold	The names of FrameMaker *User Interface* objects (menus, menu items, and buttons). The **>** symbol is used as shorthand notation for navigating to menu items and sub menus. For example, **Element > Validate...** refers to the **Validate...** item in the **Element** menu.

Using other FrameMaker documentation

The *Using FrameMaker* makes up the primary end-user documentation for this product. It explains how to use the FrameMaker authoring environment for both structured and unstructured documents. It also explains how to create templates for your documents.

In creating a structured template, you can refer to this manual for information on how your end user interacts with the product and how to create a formatted template.

New features and changes in release 12(including those for structure applications and structured documents) are listed and briefly described in the *FrameMaker Getting Started Guide*.

You will also find a range of other online documents from the FrameMaker help and support page, http://www.adobe.com/support/framemaker/.

Using FDK manuals

If you create an XML and SGML API client for your XML or SGML application, you'll need to be familiar with the FDK. FDK documentation is written for developers with C programming experience.

- *FDK Programmer's Guide* is your manual for understanding FDK basics. This manual describes how to use the FDK to enhance the functionality of FrameMaker and describes how to use the FDK to work with structured documents. To make advanced modifications to the software's default translation behavior, refer to the *Structure Import/Export API Programmer's Guide*.)
- *FDK Programmer's Reference* is a reference for the functions and objects described in the *FDK Programmer's Guide*.
- *Structure Import/Export API Programmer's Guide* explains how to use the FDK to make advanced modifications to the software's default behavior for translation between markup documents and FrameMaker documents. This manual contains both descriptive and reference information.

For information on other FDK manuals, see "Using Frame Developer Tools" in the *FDK Programmer's Guide*.

1 Structure Application Definition Reference

This chapter provides a comprehensive reference for all application properties that can be defined in a structure application definition file.

Contents of an application definition file

The highest-level element in an `structapps.fm` file is `StructuredSetup`. That element's first child must be `Version`, to indicate the FrameMaker version. The `Version` element is followed by zero or more `SGMLApplication` or `XMLApplication` elements, each of which defines the pieces of a structure application. Finally, there can be an optional `Defaults` element, which specifies information used unless overridden for a particular application.

The following table lists the main elements allowed in `structapps.fm` as children of the `StructuredSetup` element. The table identifies the sections that discuss each of those elements and the elements they may contain.

Element	Discussed in
`ApplicationName`	"Define an application," next
`SGMLApplication`	"Define an application" on page 9
`XMLApplication`	"Define an application" on page 9
`Defaults`	"Providing default information" on page 11

Define an application

FrameMaker collects all information pertaining to the set-up of a structured application into an `SGMLApplication` or `XMLApplication` element. These elements have one required child element and several optional child elements.

The first child of a parent `SGMLApplication` or `XMLApplication` element must be `ApplicationName` and gives the name of the application. It looks like:

Application name: *`name`*

where *`name`* is a string used to identify your application in the **Set Structure Application** and **Use Structure Application** dialog boxes. You cannot use the same name for multiple structure applications.

If present, the optional child elements can occur in any order and can include the following elements, discussed in the named sections:

Element	Discussed in
DOCTYPE	"Specifying a DOCTYPE element" on page 13
DTD	"Specifying a DTD" on page 14
CharacterEncoding	"Specifying the character encoding for SGML files" on page 11
ConditionalText	"Specifying conditional text output" on page 13
Entities	"Specifying entities" on page 14
ExternalXRef	"Specifying external cross reference behavior" on page 20
FileExtensionOverride	"Specifying filename extensions" on page 21
Namespace	"Enabling namespaces" on page 22
ReadWriteRules	"Specifying a read/write rules document" on page 22
RulesSearchPaths	"Specifying a search path for included files in rules documents" on page 22
Schema	"Specifying a Schema for XML" on page 23
SGMLDeclaration	"Specifying an SGML declaration" on page 24
Stylesheets	"Managing CSS import/export and XSL transformation" on page 24
Template	"Specifying a FrameMaker template" on page 27
UseAPIClient, UseDefaultAPIClient,	"Specifying a structure API client" on page 27
XMLDisplayEncoding	"Specifying the character encoding for XML files" on page 27
XMLExportEncoding	"Exporting XML" on page 30
XMLCharacterEncoding	"Specifying the character encoding for SGML files" on page 11
XMLWriteRules	"Specifying the character encoding for SGML files" on page 11
FormView	"Defining a Form view for the structured application" on page 31
MathML	"Specifying MathML options" on page 32

Some elements provide pathnames (for entities and read/write rules files; hence RulesSearchPaths and EntitySearchPaths elements). If the pathname is absolute, the software looks there. If it can't find it via the specified path, the log reports an error and the operation is aborted. If a relative pathname is given, the software looks for the file in several places:

- The directory containing the file being processed. For example, if you're opening a DTD, the software first searches the directory in which it found the DTD.
- $STRUCTDIR (for information on what directory this is, see Developer Guide, page 131: Location of structure files).

- The directory from which you started FrameMaker.

If an application definition includes any of these elements, the value in the application definition overrides any value for that element in the `Defaults` element. The sections following the next section describe these elements in detail.

Providing default information

Some of the information you provide for individual applications may be common to all your applications. For such information you can specify defaults that are used whenever an application does not provide its own version of the information. You use the `Defaults` element to provide such information.

If present, the optional child elements of `Defaults` can occur in any order (with the exception of the `Graphics` element, which must be the last child) and can include the following elements, which are discussed in the named sections:

Element	Discussed in
`CharacterEncoding`	"Specifying the character encoding for XML files" on page 27
`DTD`	"Specifying a DTD" on page 14
`Entities`	"Specifying entities" on page 14
`FrameDefaultAPIClient`, `UseAPIClient`	"Specifying a structure API client" on page 27
`MaxErrorMessages`	"Limiting the length of a log file" on page 30
`Namespace`	"Enabling namespaces" on page 22
`ReadWriteRules`	"Specifying a read/write rules document" on page 22
`RulesSearchPaths`	"Specifying a search path for included files in rules documents" on page 22
`SGMLDeclaration`	"Specifying an SGML declaration" on page 24
`Stylesheets`	"Managing CSS import/export and XSL transformation" on page 24
`Template`	"Specifying a FrameMaker template" on page 27
`XMLCharacterEncoding`	"Specifying the character encoding for SGML files" on page 11
`XMLWriteRules`	"Specifying the character encoding for XML files" on page 27
`Graphics`	"Mapping graphic notations to file types" on page 31

Specifying the character encoding for SGML files

The `CharacterEncoding` element tells the software which encoding to use for the SGML text. Typically, this is only important on non-Western systems, or in SGML applications that encounter SGML files using double-byte text. It can contain one of the following child elements:

`ISOLatin1`, `ASCII`, `ANSI`, `MacASCII`, `ShiftJIS`, `KSC8EUC`, `GB8EUC`, `CNSEUC`, `Big5`, `JIS8EUC`. The `CharacterEncoding` element looks like this:

SGML character encoding: Iso Latin1

On a non-Western system, the text for an SGML file can contain double-byte text. This text can be in any one of a number of different text encodings.

FrameMaker can interpret SGML files that contain double-byte text in *#PCDATA*, *RCDATA*, and *CDATA*. The software expects all other text to be within the 7-bit ASCII range (which is supported by all Asian fonts). This means that document content can be in double-byte encodings, but the markup must be in the ASCII range. Typically, for example, the only text in a DTD that will contain double-byte characters would be text used to specify attribute values.

> **Important:** For SGML documents, you should not use accented characters in element tag names nor attribute names. If you use such characters, FrameMaker may not be able to correctly import or export the document.

To import and export SGML that contains double-byte text, you should specify the character encoding to use, either as a default for all applications, or for a specific SGML application. For a given SGML application there can only be one encoding. If you don't specify an encoding for your application, FrameMaker determines the encoding to use by considering the current default user interface language and the current operating system; for the current language, it uses the operating system's default encoding. The default encodings for Windows® are:

Languages	Windows
Roman languages	ANSI
Japanese	Shift-JIS
Simplified Chinese	GB8 EUC
Traditional Chinese	Big5
Korean	KSC8 EUC

You can have an Asian language for the user interface, but the content of the document files in Roman fonts. In this case, any exported Roman text that falls outside of the ASCII range will be garbled. For this reason, we recommend that you specify an encoding for any application that might be used on a non-Western system.

The template for your application must use fonts that support the language implied by the encoding you specify. Otherwise, the text will appear garbled when imported into the template. You can fix this problem after the fact by specifying different fonts to use in the resulting files.

Specifying conditional text output

Add a `ConditionalText` child to the `XMLApplication` element to control conditional text output. Place a single child, `OutputTextPI` in this element. Then add one of the four children listed in the following table to the `OutputTextPI` element:

Child of OutputTextPi	FrameMaker outputs hidden conditional text	Processing instructions delimit conditional text
`OutputAllTextWithPIs`	yes	yes
`OutputAllTextWithoutPIs`	yes	no
`OutputVisibleTextWithPIs`	no	yes
`OutputVisibleTextWithoutPIs`	no	no
`OutputAllTextWithPIsFiltered`	yes	yes*
`OutputVisibleTextWithPIsFiltered`	no	yes*

*PIs are displayed only if the document settings are different from the template settings.

The `ConditionalText` element can only be a child of an `XMLApplication` element.

Specifying a DOCTYPE element

The `DOCTYPE` element specifies the generic identifier of the DOCTYPE declaration and root element in markup documents used with this application. If you open a markup document with the matching document element specified in the DOCTYPE declaration, FrameMaker uses this application when translating the document. The element looks like:

DOCTYPE: *`doctype`*

where *`doctype`* identifies a document element.

For example,

DOCTYPE: chapter

matches a markup document with the following declaration:

```
<!DOCTYPE chapter ...>
```

If more than one application defined in the `structapps.fm` file specifies the same document element, and the end user opens a file with that document element, the software gives the user a choice of which of these applications to use. If the user opens a markup document for which no application specifies its document element, the software gives the user the choice of all defined applications.

You can use more than one DOCTYPE element for an application, if that application is applicable to multiple document elements. For example, if the Book application applies when the document element is either chapter or appendix, you can use this definition:

Application name: Book
DOCTYPE: chapter
appendix
...

The DOCTYPE element can be a child of an SGMLApplication or XMLApplication element.

Specifying a DTD

The DTD element specifies a file containing the external DTD subset that FrameMaker uses when importing and exporting a markup document. It looks like:

DTD: *dtd*

where *dtd* is the pathname of a file containing a document type declaration subset.

Note that the file you specify with the DTD element must be an external DTD subset. It cannot be a complete DTD. That is, the file cannot have the form:

```
<!DOCTYPE book [
   <!element book . . .>
   . . .
]>
```

Instead, it should simply have the form:

```
<!element book . . .>
. . .
```

For more information on external DTD subsets, see Developer Guide, page 89: XML and SGML DTDs.

You can have only one DTD element for each SGMLApplication or XMLApplication. It can also be a child of the Defaults element.

Specifying entities

To specify the location of various entities, you use the Entities element. It looks like this:

Entity locations

The possible child elements of a parent Entities element are:

Element	Discussed in
EntityCatalogFile	"Specifying entities through an entity catalog" on page 15

Element	Discussed in
Entity	"Specifying the location of individual entities" on page 16
FileNamePattern	"Specifying names for external entity files" on page 17
Public	"Specifying public identifiers" on page 18
EntitySearchPaths	"Specifying a search path for external entity files" on page 19

If you use the EntityCatalogFile element, you cannot use any of the elements Entity, FilenamePattern, or Public.

You can have only one Entities element for each application, although that Entities element can have more than one of some of its child elements. The Entities element can also be a child of the Defaults element.

Specifying entities through an entity catalog

The EntityCatalogFile element specifies a file containing mappings of an entity's public identifier or entity name to a filename. It looks like:

Entity locations
Entity catalog file: *fname*

where *fname* is the filename of the entity catalog. Entity catalogs and their specified format are described below.

You can specify multiple EntityCatalogFile elements in a single Entities element. If you use this element, you cannot use any of the Entity, FilenamePattern, or Public elements.

You can use the EntityCatalogFile element both in the Entities element of the Defaults element and in an SGMLApplication or XMLApplication element to specify information for a particular application. When searching for an external entity, FrameMaker searches the application's entity catalogs before searching the default entity catalogs.

If you have an EntityCatalogFile element in an application definition, the software ignores Entity, FilenamePattern, and Public elements in the Defaults element.

Why use entity catalogs

Technical Resolution 9401:1994 published by SGML Open discusses entity management issues affecting how SGML documents work with each other:

- Interpreting external identifiers in entity declarations so that an SGML document can be processed by different tools on a single computer system
- Moving SGML documents to different computers in a way that preserves the association of external identifiers in entity declarations with the correct files or other storage objects.

The technical resolution uses *entity catalogs* and an interchange packaging scheme to address these issues. FrameMaker supports such entity catalogs with the EntityCatalogFile element.

Entity catalog format

Each entry in the entity catalog file associates a filename with information about an external entity that appears in a markup document. For example, the following are catalog entries that associate a public identifier with a filename:

```
PUBLIC "ISO 8879-1986//ENTITIES Added Latin 1//EN" "isolat1.ent"
PUBLIC "-//USA/AAP//DTD BK-1//EN" "aapbook.dtd"
```

In addition to entries mapping public identifiers to filenames, an entry can associate an entity name with a filename:

```
ENTITY "chips" "graphics\chips.tif"
```

A single catalog can contain both types of entry.

If the specified filename in a catalog entry is a relative pathname, the path is relative to the location of the catalog entry file.

For a complete description of the syntax of a catalog entry, see *Technical Resolution 9401:1994 Entity Management* published by SGML Open.

How FrameMaker searches entity catalogs

A single application may use multiple catalog files. When trying to locate a particular external entity, FrameMaker searches the files one at a time until it finds the entry it is looking for. In each file, the software first searches for an entity using the external entity's public identifier. If the software finds the identifier, it uses the associated filename to locate the entity. If it does not find the public identifier, the software searches the file looking for the entity name. If it does not find the entity name either, the software continues searching in the next catalog file.

In some circumstances, a system identifier specified in an external entity declaration may not be valid. If so, FrameMaker uses public identifier and entity name mappings.

Specifying the location of individual entities

Instead of using an entity catalog to associate entities with files, you can use the `Entity` element as a child of a parent `Entities` element. This element allows you to directly associate a filename with an individual entity. It looks like:

Entity locations
Entity name: *ename*
Filename: *fname*

where *ename* is the name of an entity and *fname* is a filename.

You can specify multiple child `Entity` elements for a single `Entities` element. You use the `FilenamePattern` and `EntitySearchPaths` elements to help the software find these files.

The `Entity` element can be a child of a parent `Entities` element in the `Defaults` element to set default entity information, and of a parent `SGMLApplication` or `XMLApplication`

element to specify information for a particular application. When searching for an external entity, the software searches the application's entity locations before searching the default entity locations.

Specifying names for external entity files

One or more `FilenamePattern` elements can appear as a child of a parent `Entities` element to tell the software how to locate an external entity.

A `FilenamePattern` element does not apply to an entity for which there is an `Entity` element. Otherwise, it applies to all external entities except those with an external identifier that includes a public identifier but no system identifier. The `FilenamePattern` looks like:

Entity locations:
Filename pattern: *`pattern`*

where *`pattern`* is a string representing a device-dependent filename. The three variables that can appear within *`pattern`* are interpreted as follows:

Variable	Interpretation
`$(System)`	The system identifier from the entity declaration
`$(Notation)`	The notation name from the entity declaration of an external data entity
`$(Entity)`	The entity name

Case is not significant in variable names, although it may be significant in the values of the variables. If a variable is undefined in a particular context, that variable evaluates to the empty string.

A parent `Entities` element can contain multiple child `FilenamePattern` elements. The software assumes the last pattern in the `Entities` element is:

Filename pattern: $(System)

Thus, if no `FilenamePattern` elements appear or even if no `Entities` element appears, the software assumes system identifiers are complete pathnames and will check search paths to locate the file.

How FrameMaker searches filename patterns

When locating an external entity, FrameMaker tests the value of the *`pattern`* arguments in successive `FilenamePattern` elements that have the same parent `Entities` element, in the order they occur, until it finds the name of an existing file. As it tests each *`pattern`*, it substitutes relevant information from the entity's declaration for variables in *`pattern`*.

You can use the `FilenamePattern` element both in the `Entities` element of the `Defaults` element and in an `SGMLApplication` element to specify information for a particular application. When searching for an external entity, FrameMaker tests all the filename patterns specified for the application before it tests those in default `FilenamePattern` elements.

Example

Suppose the Entities element looks like:

Entity locations:
Filename pattern: $(System).sgm
Filename pattern: $(System).$(Notation)

and the markup document contains:

```
<!ENTITY intro SYSTEM "introduction.xml">
<!ENTITY chips SYSTEM "chipsfile" NDATA cgm>
. . .
&intro;
. . .
<graphic entity=chips>
```

When processing the reference to intro, the software searches for a file called introduction.xml. It is an error if the file does not exist.

When processing the entity attribute of the graphic element, FrameMaker searches for a file named chipsfile.cgm. If one is not found, it then looks for chipsfile.CGM, assuming that the NAMECASE GENERAL parameter of the associated SGML declaration is NAMECASE GENERAL YES.

> **Note:** The NAMECASE GENERAL parameter of the SGML declaration determines the case-sensitivity of notation names. For XML, the implied setting for this parameter is NO, which means that names are case-sensitive.
>
> For SGML, the value of this parameter in the reference concrete syntax is NAMECASE GENERAL YES. With this declaration, the SGML parser forces notation names to uppercase.

Specifying public identifiers

The Public element of an Entities element tells the software how to process an external identifier that has a public identifier but no system identifier. It looks like:

Entity locations:
Public ID: *pid*
Filename: *fname*

where *pid* is a public identifier and *fname* is the name of a file to be associated with the entity using the public identifier.

You can give multiple Public elements in the same parent Entities element. If you want to give multiple filenames to search for a particular public identifier, you can specify the same public identifier in multiple Public elements.

You can use the Public element both in the Entities element of the Defaults element and in an Entities element of an SGMLApplication or XMLApplication element to specify information for a particular application. If a Public element occurs as a child of an SGMLApplication or XMLApplication element, that identifier is used in preference to one occurring as a child of the Defaults element.

Specifying a search path for external entity files

The EntitySearchPaths child of a parent Entities element tells the software what directories to search for the files indicated by Entity, FilenamePattern, and Public elements. It looks like:

Entity locations:
Entity search paths
1: *directory*$_1$
...
N: *directory*$_n$

where each *directory*$_i$ is a device-dependent directory name. The three variables and their abbreviations that can be used to specify a directory are as follows:

Variable	Abbreviation	Interpretation
$HOME	~	The user's home directory
$SRCDIR	.	The directory containing the document entity being processed
$STRUCTDIR		The structure directory in use (for information on what directory this is, see Developer Guide, page 131: Location of structure files)

Each *directory*$_i$ value can be an absolute pathname or relative to $SRCDIR.

How FrameMaker searches for entity files

To locate an external entity, FrameMaker searches the specified directories in the order listed. You can use the EntitySearchPaths element both in the Entities element of the Defaults element and in an XMLApplication or SGMLApplication element. When searching for an external entity, FrameMaker searches the directories named in the EntitySearchPaths element for the application before it searches those in a default EntitySearchPaths element.

An Entities element can contain only one EntitySearchPaths element. The software assumes the EntitySearchPaths element ends this way:

Entity search paths
...
N: $SRCDIR

Thus, if there is no EntitySearchPaths element, the software assumes all markup files are in the same directory.

Example

Assume the Defaults element is defined as follows:

Defaults
Entity locations:
Filename pattern: $(System).sgm
Filename pattern: $(System).$(Notation)
Entity search paths
1: $HOME
2: $SRCDIR

and the markup document contains:

```
<!ENTITY intro SYSTEM "introduction.xml">
<!ENTITY chips SYSTEM "chipsfile" NDATA cgm>
. . .
&intro;
. . .
<graphic entity=chips>
```

When processing the reference to intro, the software looks for the files:

```
$HOME/introduction.xml
$SRCDIR/introduction.xml
```

until it finds one of those files. When processing the graphic element, the software searches in order for:

```
$HOME/chipsfile.cgm
$SRCDIR/chipsfile.cgm
```

Specifying external cross reference behavior

To ensure correct resolution of external cross references in XML, use the ExternalXRef element. ExternalXRef can only be a child of XMLApplication.

Change file extension to .XML

Insert an ExternalXRef child in the XMLApplication element for the application you are developing. In this ExternalXRef element, insert a ChangeReferenceToXML child. Finally, insert an Enable element into the ChangeReferenceToXML element. It will look like this:

External X-Ref:
Change Reference To .XML: Enable

When a document with an external cross-reference is saved to XML, FrameMaker then changes the extension in the xref's `srcfile` attribute to `.xml` and exports the cross-reference as:

```
<xref srcfile="filepath/filename.xml#elemID">
```

Where:

- *`filepath`* is the absolute path to the saved source XML file
- *`filename`* is the name of the saved source XML file
- *`elemID`* is the ID of the referenced element.

You can save the source file to XML before or after saving the original file to XML. In either case, the file name specified for the XML document must be identical to the filename of the original FrameMaker document except for the extension.

If you insert a `Disable` element instead of an `Enable` element into `ChangeReferenceToXML`, FrameMaker retains the default behavior and does not change the extension in the `srcfile` attribute.

Try alternative extensions

`TryAlternativeExtensions` specifies an option for importing external cross-references from XML. It looks like this:

External X-Ref:
Try Alternative Extensions: Enable

If its content is `Enable`, and FrameMaker cannot open the file specified by the `srcfile` attribute, it changes the extension and tries to open the resulting file instead. In particular, if the original extension is `.xml`, FrameMaker also tries `.fm`; if the original extension is `.fm`, FrameMaker also tries `.xml`. If the content of `TryAlternativeExtensions` is `Disable`, FrameMaker creates an unresolved cross-reference if the specified file cannot be opened. `Disable` is the default.

Specifying filename extensions

The `FileExtensionOverride` element specifies a filename extension to use when saving a FrameMaker document as markup. This is particularly useful when saving XHTML documents. Some web browsers that support XHTML can only read files with a `.htm` or `.html` extension. When you save a document as XML (even using the XHTML doctype) FrameMaker gives the file a `.xml` extension by default. You can use this element to specify a `.htm` extension when saving a document as XHTML. The `FileExtensionOverride` element looks like this:

File Extension Override: *`extension`*

where *`extension`* is the string for the filename extension, minus the dot character. You can have only one `FileExtensionOverride` element for each XML or SGML structure application.

Enabling namespaces

The `Namespace` element specifies whether the current XML structure application supports namespaces in XML. This element can contain either an `Enable` or `Disable` child element. The `Namespace` element looks like this with namespaces enabled:

Namespace: *Enable*

You can have only one `Namespace` element for each XML structure application. It can also be a child of the `Defaults` element. It is not applicable for an SGML application.

> **Note: XML Schema:** You must enable namespaces to allow FrameMaker to validate XML against a Schema definition upon import and export. Schema allows an XML document to reference multiple Schema locations in different namespaces. When this is the case, only the first namespace is used. See Developer Guide, page 201: Schema location for additional information.

Specifying a read/write rules document

The `ReadWriteRules` element specifies the read/write rules document associated with the application. It looks like:

Read/write rules: *`rules`*

where *`rules`* is the pathname of a FrameMaker read/write rules document.

You can have only one `ReadWriteRules` element for each application. It can also be a child of the `Defaults` element.

Specifying a search path for included files in rules documents

The `RulesSearchPaths` element is analogous to the `EntitySearchPaths` element, but it pertains to additional files you include in a read/write rules document rather than to external entities referenced within a markup document. Its `Path` child elements indicate individual directories. It looks like:

Search paths for included read/write rules files:
1: *`directory`*$_1$
...
N: *`directory`*$_n$

where each *directory*$_i$ is a device-dependent directory name. The two variables and their abbreviations that can be used to specify a directory are as follows:

Variable	Abbreviation	Interpretation
$HOME	~	The user's home directory
$STRUCTDIR		The structure directory in use (for information on what directory this is, see Developer Guide, page 131: Location of structure files)

Each *directory*$_i$ value can be an absolute pathname or relative to $RULESDIR.

How FrameMaker searches for rules files

Only one RulesSearchPaths element can occur as the child of a single parent XMLApplication or SGMLApplication element or parent Defaults element. When searching for a file you include in an read/write rules document, FrameMaker searches the directories named in the RulesSearchPaths element for the application before it searches those in the RulesSearchPaths element of the Defaults element.

The software assumes RulesSearchPaths ends in this way:

Search paths for included read/write rules files:

...
N: $RULESDIR

Thus, if there is no RulesSearchPaths element, the software assumes all files you include in the read/write rules document are in the same directory as your rules document.

Specifying a Schema for XML

The Schema element, a direct child of XMLApplication, specifies the path and filename for an XML Schema file that contains element declarations for XML. It look like this:

Schema: *schema_path*

where *schema_path* is the pathname of a file containing a Schema declaration file.

In order for a structure application to be selectable in the Use Structured Application list while importing a document that is associated with a Schema, the Schema's root element must be included in the application's DOCTYPE in the XmlApplication element.

Specifying an SGML declaration

The SGMLDeclaration element specifies the location of a file containing a valid SGML declaration. It is used only for SGML applications and cannot be a child of an XMLApplication element. The SGMLDeclaration element looks like:

SGML declaration: *declaration*

where *declaration* is the pathname of the SGML declaration file.

You can have only one SGMLDeclaration element for each SGML application. It can also be a child of the Defaults element.

Managing CSS import/export and XSL transformation

The Stylesheets element of an XML structure application tells the software how to treat the use of CSS stylesheets for a given XML document, and how and whether to perform XSL transformation upon import or export of XML documents.

An XML application can have only one Stylesheets element. It can also be a child of the Defaults element.

How the Stylesheets element affects CSS generation

You can specify whether to use an existing stylesheet, or whether FrameMaker should generate a new one and use that for the exported XML. You can specify any number of stylesheets, and the exported XML will include references to each one. The Stylesheets element also contains instructions concerning the use of attributes and stylesheet processing instructions. The Stylesheets element for CSS looks like:

CSS2 Preferences:
    Generate CSS2: *enable/disable*
    Add Fm CSS Attribute To XML: *enable/disable*
Retain Stylesheet Information: *enable/disable*
XML Stylesheet:
    Type: *stylesheet_type*
    URI: *path*

When you save a document to XML, FrameMaker can either use an existing stylesheet, or generate a new one from the current EDD. How FrameMaker generates a stylesheet is determined by the values of the children of the Stylesheets element. For more information about how FrameMaker converts EDD information into a stylesheet, see Developer Guide, page 283: Saving EDD Formatting Information as a CSS Stylesheet

GenerateCSS2 Specifies whether FrameMaker will generate a CSS when you save the document as XML. It can be set to *enable* or *disable*. When this is set to *enable*, FrameMaker generates a CSS. If a path is provided in StylesheetURI, FrameMaker saves the stylesheet to that location, with

that filename. Otherwise, it saves the stylesheet to the same location as the XML document with a filename *xmldoc*.css, where *xmldoc* is the name of the XML document you're saving.

AddFmCSSAttrToXml Specifies whether FrameMaker will write instances of the fmcssattr attribute to elements in the XML document. It can be set to enable or disable. An EDD can include context selectors as criteria to assign format rules. CSS has no equivalent to this. When this is set to enable, FrameMaker uses the fmcssattr attribute in certain elements so the CSS can achieve the same formatting as the EDD.

RetainStylesheetPIs Specifies whether FrameMaker will retain the stylesheet declaration for import and export of XML. It can be set to *enable* or *disable*. When this is set to *enable*, FrameMaker does the following:

- On import, it stores the XML document's stylesheet PI as a marker in the FrameMaker document.
- On export, it writes the content of stylesheet PI marker in the resulting XML document.

StylesheetType Specifies the type of stylesheet. It contains a string for the stylesheet type. Currently, you can specify CSS (upper or lower case) or XLS (upper or lower case). If you specify XLS, FrameMaker will not generate a stylesheet.

StylesheetURI Specifies the URI for the stylesheet. It contains a string; for example, /$STRUCTDIR/xml/xhtml/app/xhtml.css.

How the Stylesheets element affects CSS import

You can specify whether a CSS stylesheet that is referenced in an XML file is used to update the formatting of the FrameMaker document. The ProcessStylesheetPI is an optional child of the CssPreferences element and looks like this:

CSS2 Preferences:
ProcessStylesheetPI: *enable/disable*

ProcessStylesheetPI can have one of the following values: Enable or Disable. If the value of the ProcessStylesheetPI element is Enable, then the CSS file referenced in the XML file is used while opening the XML file. The default value of the ProcessStylesheetPI element is Disable.

For more information about how the CSS file mentioned in the XML file is used when an XML file is opened, see Chapter 6, "CSS to EDD Mapping."

How the Stylesheets element affects XSL transformation

If an XML structure application specifies an XSL stylesheet, FrameMaker can apply transformations defined in that stylesheet when importing an XML document, or when exporting a FrameMaker document to XML. The XSLTPreferences element in the Stylesheets element allows you to specify the XSL file to use for transformation upon import (PreProcessing), export (PostProcessing), and smart paste (SmartPaste). StylesheetParameters elements

allow you to set parameters of an XSL stylesheet at run time, before the transformation takes place.

XSLT Preferences:
- **Process Stylesheet PI:** enable/disable
- **Preprocessing:**
  - **Stylesheet:** *path*
  - **Processor:** *processor name*
  - **Stylesheet Parameters**
    - **Name:** *parameter name*
    - **Expression:** *exp*
- **Postprocessing:**
  - **Stylesheet:** *path*
  - **Processor:** *processor name*
  - **Stylesheet Parameters**
    - **Name:** *parameter name*
    - **Expression:** *exp*
- **SmartPaste:**
  - **Stylesheet:** *path*
  - **Processor:** *processor name*
  - **Stylesheet Parameters**
    - **Name:** *parameter name*
    - **Expression:** *exp*

ProcessStylesheetPI Specifies whether FrameMaker will use the XSL file mentioned in the `xml-stylesheet` PI of an XML file to transform that file. It can be set to `enable` or `disable`. By default it is set to `disable`, and FrameMaker does not use the PI. Set to `enable` to use the PI.

PreProcessing Contains a `Stylesheet` element that specifies the XSL file to be used for transformation upon import of an XML document. Transformation occurs before read rules are applied. The `XSLTPreferences` element can contain 0 or 1 `PreProcessing` elements.

PreProcessing Contains a `Stylesheet` element that specifies the XSL file to be used for transformation upon export of an XML document. Transformation occurs after write rules are applied. The `XSLTPreferences` element can contain 0 or 1 `PostProcessing` elements.

SmartPaste Contains a `Stylesheet` element that specifies the XSL file to be used for transformation upon pasting content from an external application.

Stylesheet Specifies the URI for the XSL file. It contains a string; for example, `/$STRUCTDIR/xml/xhtml/app/mystyles.xsl`.

Processor Specifies the processor for the XSL file. You can either choose from the existing options - SAXON or XALAN, or specify your own processor name.

StylesheetParameters Contains `ParameterName` and `ParameterExpression` pairs. Each pair specifies the name of a parameter used the XSL stylesheet, and an expression that constrains the value of that parameter for the subsequent transformation.

For more information on XSL transformation of XML, see Developer Guide, Chapter 29, Additional XSL Transformation for XML.

Specifying a FrameMaker template

The `Template` element specifies the location of the FrameMaker template. It looks like:

Template: *`template`*

where *`template`* is the pathname of a FrameMaker template.

The software uses this template to create new FrameMaker documents from markup documents, which may be single documents resulting from the `Open` or `Import` command or documents in a book created through the `Open` command.

If this element is not present, the software creates new portrait documents as needed. When you import a markup document into an existing document, the software uses the import template only to access reference elements that are stored on the template's reference page. (For information about reference elements, see Developer Reference, page 332: Translating SDATA entities as FrameMaker reference elements.)

You can have only one `Template` element for each application. It can also be a child of the `Defaults` element.

Specifying a structure API client

In an application definition, the `UseDefaultAPIClient` element tells the software that your application does not use a special client for markup translation. In the defaults section, the `FrameDefaultAPIClient` element serves the same purpose. The default client is named `FmTranslator`.

If you do need a structure API client, use the `UseAPIClient` element in either context. For information on creating structure API clients for a structure application, see the online manual *Structure Import/Export API Programmer's Guide*.

Specifying the character encoding for XML files

The XML specification supports UNICODE characters for document content and markup tokens. In XML the given encoding is specified in the document prolog. The following example shows a specification for ShiftJIS character encoding:

```
<?xml version="1.0" encoding="Shift_JIS" ?>
```

The XML specification states that an XML document must either specify an encoding in the prolog, or it must be UTF-8 or UTF-16. FrameMaker follows this specification by assuming UTF-8 by default if there is no encoding specified in the XML file.

If you read an XML file with character encoding that does not match either the declared encoding or the default encoding (if no encoding is declared), it is likely that the import process will encounter a character that does not match the encoding FrameMaker uses. In that case, you will get a parsing error that says the document is not well-formed due to a bad token.

FrameMaker uses the encoding statement in the document prolog to determine which encoding to use. The statement must specify one of the encodings supported by your specific FrameMaker installation. FrameMaker ships with support for the following encodings:

Big5	KSC_5601
EUC-JP	Shift_JIS
EUC-KR	US-ASCII
EUC-TW	UTF-16
GB2312	UTF-8
ISO-8859-1	windows-1252

You can add other encodings to your FrameMaker installation—see Developer Guide, page 103: Unicode and character encodings.

FrameMaker converts the encoding of the XML document to an internal display encoding. In this way FrameMaker fully supports Unicode characters for text that is in *#PCDATA*, *RCDATA*, and *CDATA*. For any #PCDATA character that it cannot interpret, FrameMaker uses a marker of type `UNKNOWNCHAR` to represent the character. For unknown CDATA characters, FrameMaker uses XML character references.

The following sections describe how to control the display encoding that FrameMaker uses, and how to specify an encoding when you save a document as XML.

Display encoding

On import, FrameMaker converts the XML encoding to a display encoding that is appropriate for a given language. However, FrameMaker cannot automatically determine which conversion to make. Although the XML document prolog specifies an encoding, the document may contain elements or other constructs that override the language implied by that encoding. As a result, you should specify a display encoding for the structure application. The display encodings you can specify are:

Display encoding:	**For this language:**
FrameRoman	Western European languages
JISX0208.ShiftJIS	Japanese
BIG5	Traditional Chinese
GB2312-80.EUC	Simplified Chinese
KSC5601-1992	Korean

By default, FrameMaker uses the display encoding that matches the locale of your operating system. To specify a different display encoding, use the `XmlDisplayEncoding` element. `XmlDisplayEncoding` can contain one child element to specify one of the supported display encodings.

The display encoding also determines how FrameMaker interprets the characters in markup tokens such as GIs and attribute names. If FrameMaker encounters such a token with an unknown character, FrameMaker drops the token. For more information, see Developer Guide, page 101: Supported characters in element and attribute names.

For example, if your operating system locale is French, German, or English FrameMaker uses FrameRoman by default. This is true, even if the XML prolog specifies an encoding for a different language, such as ShiftJIS. To import XML encoded as ShiftJIS, you would use the `XmlDisplayEncoding` element to specify JISX0208.ShiftJIS, as follows:

XML Display Encoding: JISX0208.ShiftJIS

When you specify such an encoding, FrameMaker uses that encoding as the default for all the *#PCDATA*, *RCDATA*, and *CDATA* in the imported XML. Markup tokens that include characters in the upper range of the display encoding are interpreted correctly. If you have fonts installed for the display encoding, then the text will appear as intended.

For another example, assume you have a version of US English FrameMaker installed on a Traditional Chinese operating system. By default, FrameMaker uses Big5 as the display encoding. It also supports any Big5 characters that are used in GIs and attribute names. If you are importing an XML document that is in English, you would need to specify FrameRoman as the display encoding.

Note that the XML standard includes the `xml:lang` attribute. This attribute can specify a change of language for an element and its content. If that language is one of those listed in the table of display encodings, a change made by this attribute take precedence over the setting made via `XmlDisplayEncoding`.

Finally, the template for your application must use fonts that support the given language. Otherwise, the text will appear garbled when imported into the template. You can fix this problem by specifying different fonts to use in the resulting files.

Encoding of CSS files

FrameMaker supports the following encodings for CSS files: utf-8, utf-16, utf-16LE, and utf-16BE. FrameMaker detects the encoding of a CSS file using the Byte Order Mark (BOM), and not the "@charset" statement.

Exporting XML

Your XML structure application can include an `XmlExportEncoding` element to specify the encoding to use when you save a document as XML. FrameMaker determines which encoding to use according to the following rules:

If:		FrameMaker uses:
1.	The structure application specifies a value for `XmlExportEncoding`, and that encoding is supported	The specified encoding
2.	1 is not true, and the original XML source specified an encoding, and that encoding is supported	The encoding that was specified in the original XML source
3.	1 and 2 are not true	UTF-8

The `XmlExportEncoding` element contains a string for the name of an encoding. The name you provide must conform with the IANA naming conventions. The standard installation of FrameMaker supports the encodings that are listed at the beginning of this discussion (see page 27).

For example, to export your document as ISOLatin1, use the `XmlExportEncoding` element as follows:

XML Export Encoding: ISO-8859-1

Limiting the length of a log file

The `MaxErrorMessages` child element of the `Defaults` element allows you to limit the length of structure error reports. It looks like:

Maximum number of error messages: *n*

where *n* is the desired limit. If *n* is less than 10, the software resets it to 10. This must be the last child of the parent `Defaults` element.

By default, FrameMaker does not write more than 150 messages (error messages and warnings) to a single log file.

Messages pertaining to opening and closing book components are not included in this limit. Messages generated through your own structure API client are also not counted, although if you wish, you can count them using your own code.

In documents that generate large numbers of messages, the 151st message is replaced with a note that additional messages have been suppressed.

Note that processing continues, even though further messages are not reported. This message limit is reset for every file processed and for each component of a book.

Mapping graphic notations to file types

The `Graphics` child element of the `Defaults` element allows you to provide mappings from graphic notation to file type by using the file name extension. In the example below the `JPEG` notation is mapped to the `.jpg` extension.

Graphics
Notation: JPEG **Filetypehint:**jpg

The Graphics element may contain one or more Mapping elements.

Defining a Form view for the structured application

When you create a structured application, you can define a Form view for the application. Your authors can then use FrameMaker's Simplified XML interface. The new simplified authoring interface provides a form-like easy-to-fill authoring environment. For more information, see the *Simplified XML section* of the FrameMaker User Guide.

Form View
Configuration File: *<Path to configuration (.ini) file>*
Example: $STRUCTDIR\xml\DITA_1.2\app\FrameMaker\simplifiedxml\config\topic_config.ini
Template: *<Path to Simplified XML template file>*
Example:
$STRUCTDIR\xml\DITA_1.2\app\FrameMaker\simplifiedxml\template\topic.template.simplifiedxml.fm

Template:

When you create a structured application, you create a template file that is associated with the application. In the structured application template file, you define how to display the fields and specify the auto-insertion rules. The auto-insertion rules are used to decide the default elements. The auto-inserted elements are the elements that display when an author creates a new document. You can use the structured application tem-plate file for the Simplified XML form view. Alternatively, you can edit the auto-insertion rules change elements that display when an author creates a new document in the Form view. However, the author will still be able to add these elements to the structured document unless you exclude these elements using the configuration file.

The Form view template is similar to the structured application template. You are recommended to customize your structured application template as per the requirements of the Simplified view. If you plan to display the same default elements in the Simplified view as the WYSIWYG view, you can use the same template file. Alternatively, you can exclude the Template element in the Form View construct of the structapps.fm file. If the Template element is not found in the structapps.fm file, FrameMaker defaults to the structured application template.

Configuration File:

Configure the Simplified view using the configuration file. This file provides options to specify the structured application elements to display as form fields in the Simplified view. The paragraph format defined in the template that is used to display the form labels.

When specifying the following flags in the configuration file, you need to ensure that the corresponding tags are defined in the Simplified XML template file:

- FormLabelPgfFormat
- RequiredFormLabelPgfFormat
- FormFieldColor
- RequiredFormFieldColor
- SelectedFormFieldColor

For details on the configuration options, see the *Customize the Simplified XML authoring environment* section of the FrameMaker user guide.

To ensure that an element does not display in the form view, you can exclude the element from the auto-insertion rules defined in the template file and from the definition in the configuration file.

> **Note:** If you exclude an element using the configuration file but the element is included in the auto-insertion rules, the element will display disabled (un-editable) if the element is not empty.

Specifying MathML options

Your structured application can include a `MathML` element to specify a namespace prefix for MathML elements, and/or whether to save MathML equations as entities or Hex values. The `MathML` element looks like:

MathML
Namespace Prefix: *`nsprefix`*
Export Entities As Values: *`true`*

where *`nsprefix`* is the namespace prefix that is used for all MathML elements. The *`Export Entities As Values`* is set to *`true`* by default, which signifies that all MathML equations are saved as Hex values. If you set it to *`false`*, then MathML equations are saved as entities.

You can have only one `MathML` element for each XML structure application. It can also be a child of the `Defaults` element. It is not applicable for an SGML application.

2 Read/Write Rules Summary

This chapter lists the available read/write rules by category and briefly describes the purpose of each rule. The categories, which are arranged alphabetically, are as follows:

- "All Elements" on page 33
- "Attributes" on page 34
- "Books" on page 35
- "Cross-references" on page 35
- "Entities" on page 36
- "Equations" on page 36
- "Footnotes" on page 37
- "Graphics" on page 37
- "Markers" on page 38
- "Processing instructions" on page 39
- "Markup documents" on page 39
- "Tables" on page 40
- "Text" on page 41
- "Text insets" on page 41
- "Variables" on page 41.

All Elements

To	Use this rule	Page
Translate a markup element	`element`	56
Discard or unwrap a FrameMaker element on export	`fm element`	77
Translate a markup element to a FrameMaker element	`is fm element`	110
Translate a markup attribute within the context of a single markup element	`attribute`	46

To	Use this rule	Page
Inform FrameMaker not to update a FrameMaker element's definition when updating an existing EDD	`preserve fm element definition`	148
Discard a FrameMaker or markup element	`drop`	53
Discard the content but not the structure of a FrameMaker or markup element	`drop content`	55
Discard the structure but not the content of a markup or FrameMaker element	`unwrap`	162

Attributes

To	Use this rule	Page
Translate a markup attribute	`attribute`	46
Discard a FrameMaker attribute	`fm attribute`	76
Translate a markup attribute to a FrameMaker attribute	`is fm attribute`	104
Translate a markup attribute within the context of a single markup element	`element`	56
Discard a markup or FrameMaker attribute	`drop`	53
Translate a markup attribute to a particular FrameMaker property	`is fm property`	116
Translate a value for a markup attribute to a FrameMaker property value	`is fm property value`	126
Translate a value of a markup notation attribute or name token group to a value for a FrameMaker choice attribute	`is fm value`	139
Translate a markup attribute value to a FrameMaker property or a choice attribute value	`value`	165
Specify the value to use for a markup implied attribute when a document instance provides no value	`implied value is`	97

Books

To	Use this rule	Page
Specify whether to use elements or processing instructions to indicate book components when reading a markup document	`generate book`	93
Specify elements to use to indicate book components when reading a markup document	`put element` (described with `generate book`)	93
Specify the use of processing instructions to indicate book components when reading a markup document	`use processing instructions` (described with `generate book`)	93
Specify whether or not to write processing instructions that indicate book components in a markup document	`output book processing instructions`	147

Cross-references

To	Use this rule	Page
Translate markup elements to FrameMaker cross-reference elements	`is fm cross-reference element`	109
Translate FrameMaker cross-reference properties when no markup attribute exists	`fm property`	80
Translate FrameMaker cross-reference properties when no markup attribute exists	`value is` (described with `fm property`)	80
Translate a markup attribute to a particular FrameMaker property	`is fm property`	116
Translate a value for a markup attribute to a FrameMaker property value	`is fm property value`	126
Translate a value of a markup notation attribute or name token group to a value for a FrameMaker choice attribute	`is fm value`	139
Translate a FrameMaker cross-reference element to text in markup	`fm element unwrap`	77, 162

Entities

To	Use this rule	Page
Translate a markup entity reference to an appropriate FrameMaker representation	`entity`	61
Determine the form of names of entities created for exported graphics	`entity name is`	63
Drop references to external data entities	`external data entity reference`	72
Translate an entity reference to a FrameMaker variable	`is fm variable`	140
Translate an entity reference to a single character	`is fm char`	107
Translate an entity reference to an element on a reference page	`is fm reference element`	128
Translate an SDATA entity reference to a text inset	`is fm text inset`	136
Determine the formatting of a text inset	`reformat as plain text`	154
	`reformat using target document catalogs`	155
	`retain source document formatting`	156
Discard external data entity references	`drop`	53

Equations

To	Use this rule	Page
Translate a markup element to a FrameMaker equation element	`is fm equation element`	111
Specify export information for translating FrameMaker equations	`equation`	65
Specify the filename used for exporting an equation	`export to file`	69
Determine the form of names of entities created for exported equations	`entity name is`	63
Specify the data content notation for an exported equation	`notation is`	145

To	Use this rule	Page
Determine whether FrameMaker uses the `dpi` attribute or the `impsize` attribute for equations and also the resolution used	`specify size in`	157
Translate FrameMaker cross-reference properties when no markup attribute exists	`fm property`	80
Translate FrameMaker cross-reference properties when no markup attribute exists	`value is` (described with `fm property`)	80
Translate FrameMaker equation properties to markup attributes	`is fm property`	116
Translate a value for a markup attribute to a FrameMaker property value	`is fm property value`	126
Translate a value of a markup notation attribute or name token group to a value for a FrameMaker choice attribute	`is fm value`	139
Translate a markup attribute value to a FrameMaker property or a choice attribute value	`value`	165

Footnotes

To	Use this rule	Page
Translate a markup element to a FrameMaker footnote element	`is fm footnote element`	113

Graphics

To	Use this rule	Page
Translate a markup element to a FrameMaker graphic element	`is fm graphic element`	114
Specify export information for translating FrameMaker graphics	`anchored frame`	43
Specify export information for translating FrameMaker graphics that have a single inset	`facet`	74
Specify the filename used for exporting a graphic or a facet of a graphic	`export to file`	69
Force the software to export graphic files that were imported by reference	`convert referenced graphics`	51

To	Use this rule	Page
Determine the form of names of entities created for exported graphics	`entity name is`	63
Specify the data content notation for an exported graphic	`notation is`	145
Determine whether FrameMaker uses the `dpi` attribute or the `impsize` attribute for imported graphics objects and also the resolution used	`specify size in`	157
Translate FrameMaker cross-reference properties when no markup attribute exists	`fm property`	80
Translate FrameMaker cross-reference properties when no markup attribute exists	`value is` (described with `fm property`)	80
Translate FrameMaker graphic properties to markup attributes	`is fm property`	116
Translate a value for a markup attribute to a FrameMaker property value	`is fm property value`	126
Translate a value of a markup notation attribute or name token group to a value for a FrameMaker choice attribute	`is fm value`	139
Translate a markup attribute value to a FrameMaker property or a choice attribute value	`value`	165

Markers

To	Use this rule	Page
Discard FrameMaker non-element markers or translate them to processing instructions	`fm marker`	79
Translate a markup element to a FrameMaker marker element	`is fm marker element`	115
Determine whether marker text for marker elements becomes content or an attribute value in markup	`marker text is`	143
Drop references to external data entities	`external data entity reference`	72
Drop unrecognized processing instructions	`processing instruction`	151
Translate FrameMaker non-element markers to processing instructions	`is processing instruction`	141
Discard non-element markers	`drop`	53

To	Use this rule	Page
Translate FrameMaker cross-reference properties when no markup attribute exists	`fm property`	80
Translate FrameMaker cross-reference properties when no markup attribute exists	`value is` (described with `fm property`)	80
Translate FrameMaker marker properties to markup attributes	`is fm property`	116
Translate a value for a markup attribute to a FrameMaker property value	`is fm property value`	126
Translate a value of a markup notation attribute or name token group to a value for a FrameMaker choice attribute	`is fm value`	139
Translate a markup attribute value to a FrameMaker property or a choice attribute value	`value`	165

Processing instructions

To	Use this rule	Page
Specify the treatment of unrecognized processing instructions	`processing instruction`	151
Specify the use of processing instructions to indicate book components when reading a markup document	`use processing instructions` (described with `generate book`)	93
Specify whether or not to write processing instructions that indicate book components in a markup document	`output book processing instructions`	147
Translate FrameMaker non-element markers to specific markup, or drop them	`fm marker`	79
Translate FrameMaker non-element markers to processing instructions	`is processing instruction`	141
Discard processing instructions	`drop`	53

Markup documents

To	Use this rule	Page
Specify whether or not to use an external DTD subset to contain the DTD for a markup document created by FrameMaker	`include dtd`	98

To	Use this rule	Page
Specify whether or not to include an SGML declaration in an SGML document created by FrameMaker	`include sgml declaration`	100
Specify the system and public identifiers for an external DTD subset	`external dtd`	73
Specify whether to create an entire markup document or just a markup document instance	`write structured document`	166
	`write structured document instance only`	167

Tables

To	Use this rule	Page
Translate a markup element to a FrameMaker table element	`is fm table element`	133
Translate a markup element to a FrameMaker element for a particular table part	`is fm table part element`	135
When creating a FrameMaker table, insert a table part even if that part is empty	`insert table part element`	101
Specify that a particular element always indicates a new table row	`start new row`	159
Indicate the start of a vertical straddle	`start vertical straddle`	160
Indicate the end of a vertical straddle	`end vertical straddle`	59
Specify the ruling style used for all tables	`table ruling style is`	161
Specify the resolution used for column widths with proportional widths	`proportional width resolution is`	152
Specify that the software write the width of table columns using proportional units	`use proportional widths`	164
Translate FrameMaker table properties to markup attributes	`is fm property`	116
Translate a value for a markup attribute to a FrameMaker property value	`is fm property value`	126
Translate a value of a markup notation attribute or name token group to a value for a FrameMaker choice attribute	`is fm value`	139
Translate a attribute's name token value to a FrameMaker property or choice value	`value`	165

Text

To	Use this rule	Page
Translate an entity reference to a single character	`is fm char`	107
Determine the treatment of line breaks in reading and writing markup documents	`line break`	142
Define mappings between characters in the markup and FrameMaker character sets	`character map`	49

Text insets

To	Use this rule	Page
Translate an SDATA entity reference to a FrameMaker text inset	`entity`	61
	`is fm text inset`	136
Determine the formatting of a text inset	`reformat as plain text`	154
	`reformat using target document catalogs`	155
	`retain source document formatting`	156

Variables

To	Use this rule	Page
Translate a markup element to a FrameMaker system variable element	`is fm system variable element`	132
Translate an entity reference to a FrameMaker variable	`is fm variable`	140
Translate a markup entity reference to a FrameMaker variable	`entity`	61
Determine treatment of FrameMaker non-element variables	`fm variable`	92
Translate a FrameMaker system variable element to text in markup	`fm element unwrap`	77, 162
Discard nonelement variables	`drop`	53

3 Read/Write Rules Reference

This chapter provides a reference to all read/write rules, listed in alphabetical order. The entry for each rule starts with a brief explanation of the purpose of the rule and how to use it. The rule's description may include the following sections:

Synopsis and contexts The rule's syntax and the context in which it can be used. If the rule occurs as a subrule of another rule, the more general rule is shown. If the rule can be used in multiple contexts, the synopsis shows each context. Each entry in this section shows a valid rule that has the current rule either at the highest level or as one of its subrules.

Rule synopses use the following conventions:

- Bold portions and nonitalicized portions of a rule are entered by you as shown.
- Italicized portions of a rule indicate the rule's arguments or possible subrules; you enter your values.
- Brackets [ ] indicate optional parts of a rule; the entire form within the brackets can be included or omitted.

Arguments The possible arguments to the rule. If an argument is optional, its default value is provided. Some rules have *`subrule`* as one of their arguments. In these cases, a list of possible subrules is provided. Some rule arguments allow variables. In these cases, a list of possible variables is provided.

Details Instructions on how to use the rule and on FrameMaker behavior when the rule is not supplied.

XSLT interaction Useful information about the relationship between FrameMaker's Read/Write rules and equivalent XSLT processing.

Examples Various examples of the rule.

See also Cross-references to other relevant information in the manual.

For information on how to create a Read/Write rules file and on the syntax of rules, see Developer Guide, Chapter 18, Read/Write Rules and Their Syntax

anchored frame

Use the `anchored frame` rule and its subrules to define how FrameMaker handles the content of anchored frames when writing to markup and creating a referenced graphic file. Subrules can specify base entity name, file name construction, graphic file format, notation type and unit of

measure. The rule is used when an anchored frame contains FrameMaker graphics, more than one imported graphic file, or a graphic file that has been copied into the document.

> **Note:** Use the `facet` rule for anchored frames that contain single graphic files that have been imported by reference.

Synopsis and contexts

```
1.element "gi" {
      is fm graphic element ["fmtag"];
      writer anchored frame subrule;
   . . .}
2.element "gi" {
      is fm graphic element ["fmtag"];
      writer anchored frame {
      subrules;
      }
   . . .}
```

Arguments

gi	A markup element's name (generic identifier).
fmtag	A FrameMaker element tag.
subrules	An `anchored frame` rule can have one or more of the following subrules: `entity name is`, tells the software how to create the base name for the entity associated with this element type. `export to file` tells FrameMaker how to write the file name when it creates a new graphic file, and optionally the graphic format for the file. `notation is` specifies the data content notation of the entity file. `specify size in` specifies the units to use when writing the file. `export dpi is` tells FrameMaker the dpi setting to use for the exported graphic file.

Details

The `anchored frame` rule must be a subrule of a `writer` rule for a graphic element.

On export, if the anchored frame contains only a single imported graphic file, FrameMaker uses that graphic file for the resulting markup graphic element by default. If the anchored frame contains more than one graphic file, or has been modified using FrameMaker graphics tools, the software writes out a graphic file to be used. The default format for these graphic files is CGM. The export format can be changed with the `export to file` rule. For more information about

translating anchored frame contents, see Developer Guide, Chapter 23, Translating Graphics and Equations

Examples

Assume you use the `Graphic` element for all graphic elements. If the graphic contains any single facet, assume the graphic was imported as an entity and you want the default behavior. However, if the author used FrameMaker graphic tools to create the objects in the graphic element, you want the file written in QuickDraw PICT format.

To accomplish all this, use this rule:

```
element "graphic" {
   is fm graphic element;
   writer anchored frame export to file "$(docname).pic"
   as "PICT";
}
```

Assume the FrameMaker document is named `mydoc.fm`. For the first graphic that is not a single facet, the software writes out a graphic file named `mydoc1.pic` in the PICT format.

If the export DTD declares an entity attribute to identify the graphic file with the `graphic` element, the software generates the following entity declaration:

```
<!ENTITY graphic1 SYSTEM "mydoc1.pic" NDATA PICT>
```

The corresponding graphic element in the markup could be:

```
<graphic entity = "graphic1"/>
```

If the export DTD includes only a `file` attribute to associate the graphic file with the graphic element, the software uses this filename as its value:

```
<graphic file = "mydoc1.pic"/>
```

See also

Related rules	“equation” on page 65 “facet” on page 74
Rules mentioned in synopses	“element” on page 56 “is fm equation element” on page 111 “is fm graphic element” on page 114 “writer” on page 168
General information on this topic	Developer Guide, Chapter 23, Translating Graphics and Equations

attribute

Use the `attribute` rule to describe how to process a markup attribute. By default, a markup attribute translates to a FrameMaker attribute of the same name. Usually, this rule occurs as a subrule of the `element` rule, to describe treatment of the attribute *`attr`* within the element *`gi`*.

Synopsis and contexts

```
1.[mdv] attribute "attr" {. . .
      subrule;
   . . .}
2.element "gi" {. . .
      [mdv] attribute "attr" {. . .
         subrule;
   . . .}
   . . .}
```

Arguments

`mdv`

An optional markup declared value, specifying the type of the markup attribute. Legal values for an XML application are:

- `cdata`
- `nmtoken`
- `nmtokens`
- `entity`
- `entities`
- `id`
- `idref`
- `idrefs`
- `notation`
- `group`.

Legal values for an SGML application are:

- `cdata`
- `name`
- `names`
- `nmtoken`
- `nmtokens`

- number
- numbers
- nutoken
- nutokens
- entity
- entities
- notation
- id
- idref
- idrefs
- group.

attr The name of a markup attribute.

gi A markup element's name (generic identifier).

subrule An attribute rule can have one of the following subrules:

drop discards the attribute. If this rule is used, no other attribute subrules may be used.

or:

is fm attribute translates a markup attribute into a FrameMaker attribute.

or:

is fm property translates a markup attribute to a FrameMaker property such as the width of columns in a table. This subrule is applicable only to cross-reference, marker, graphic, equation, table, and table part elements.

An attribute rule can also have the following subrules:

implied value is specifies the value to use for an impliable attribute for which no value is given in a document instance.

value translates one of the possible values of a markup name token, group or a notation attribute to a specific token of a FrameMaker choice attribute.

Details

- In some cases, the same attribute may occur in several markup elements and may require the same treatment for most of those occurrences. In these situations, you can use the

attribute rule at the highest level to set the default treatment of the attribute. You can then override the default in individual element rules.

- If the drop rule is used no other subrules of attribute may be used. The subrules is fm attribute, and is fm property are mutually exclusive. That is, if you use one of these rules, you cannot use the other rule.

Examples

- The following rule specifies that the sec attribute of the markup list element is in a name token group and corresponds to the attribute Security on the corresponding FrameMaker element:

```
element "list"
   group attribute "sec"
      is fm attribute "Security";
```

- Assume you have several elements that represent graphic objects. Each of them has an attribute w, representing the width of the object. Use this rule to make the width be 3 inches unless otherwise specified for a particular element:

```
attribute "w" {
   is fm property width;
   implied value is "3in";
}
```

- Assume you have an element team with an attribute color. The possible values for color are r, b, and g. To change the names of these values in the corresponding FrameMaker choice attribute, use this rule:

```
element "team" {
 attribute "color" {
      value "r" is fm value "Red";
      value "b" is fm value "Blue";
      value "g" is fm value "Green";
}}
```

See also

Related rules	"fm attribute" on page 76 "is fm attribute" on page 104
Rules mentioned in synopses	"element" on page 56
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes

character map

Use the `character map` rule to define mappings between characters in the markup and FrameMaker character sets. Many characters can be expressed using a string; others require using the appropriate integer character code.

> **Note: XML:** This read/write rule is primarily for SGML. XML can use UNICODE characters which makes this rule unnecessary.By default FrameMaker assumes UTF-8 encoding for XML import and export. If you want to use ISOLatin encoding with an XML document, then you may need to use this rule to map characters.

Synopsis and contexts

1.**character map** is $cmap_1$ [, . . ., $cmap_n$];

2.reader **character map** is $cmap_1$ [, . . ., $cmap_n$];

3.writer **character map** is $cmap_1$ [, . . ., $cmap_n$];

Arguments

$cmap_i$

A mapping between the character set used in the markup document and the FrameMaker character set. Each $cmap_i$ has one of the following forms:

```
sgmlch = fmch;
sgmlch = trap;
trap = fmch;
```

sgmlch is either a 1-character string or a character code representing a character in the markup character set. *sgmlch* can be a single character only if that character has the same character code in both the FrameMaker and markup character sets. Otherwise, you must use the integer character code.

fmch is either a 1-character string or a character code representing a character in the FrameMaker character set.

For information on how to represent character codes and special characters in strings, see Developer Guide, page 278: Strings and constants.

Details

- Some characters might be defined in only one of the two character sets. The keyword `trap` is provided for this situation. By default, FrameMaker discards trapped characters.
- The character map need not be a one-to-one mapping. If a character in the input document is mapped to multiple characters in the output character set, FrameMaker uses the output character from the *last* mapping to appear in the `character map` rule.

- If you use the `character map` rule at the highest level, do not also use it inside either a `reader` rule or a `writer` rule. If you use this rule inside a `reader` rule or a `writer` rule and also use it at the highest level, FrameMaker ignores the highest-level `character map` rule. You can only have one occurrence of this rule at the highest level.

  Similarly, the `character map` rule can appear in one `reader` rule and one `writer` rule at most. The software ignores any subsequent uses of the `character map` rule.

- If you use the `character map` rule at the highest level, its behavior is bidirectional. For example, you could have this rule:

  ```
  character map is 0x20 = 0x12;
  ```

  This rule specifies that the ISO Latin-1 space character (character code `0x20`) maps to the FrameMaker thin space character (character code `0x12`). With this rule, FrameMaker translates a thin space to a standard space when it writes a markup document. However, this rule translates *all* spaces in a markup document to thin spaces in a corresponding FrameMaker document. This is unlikely to be the desired behavior. For this reason, instead you should use this rule:

  ```
  reader character map is 0x20 = 0x12;
  ```

- By default, FrameMaker assumes that the character set your SGML documents use is ISO Latin-1. It provides a default mapping between those character sets. For details, see Chapter 12, "Character Set Mapping." For information on other character sets you can use, see Chapter 11, "ISO Public Entities."

- By default, on export FrameMaker produces a character in the SGML document for most printing characters in the corresponding FrameMaker document. FrameMaker documents occasionally include unusual characters that serve no purpose outside FrameMaker. For example, the codes `0x01` and `0x03` are nonprinting characters that represent information about the insertion point movement. On export FrameMaker traps such characters, so that they don't appear in an exported SGML document.

  Similarly, on import FrameMaker produces a character in the FrameMaker document for most printing characters. It traps all control characters other than a tab or newline character.

- FrameMaker has an 8-bit character set. The SGML declaration can specify any character set that the SGML parser can handle. Part of the character set description in the SGML declaration is not human-readable and may not be interpretable automatically, therefore, any differences between the native FrameMaker character set and the character set in the SGML document must be specified with the `character map` rule.

- By default, FrameMaker discards trapped characters. You can provide a structure API client to change the processing of trapped characters. For information on creating a structure API client, see the *Structure Import/Export API Programmer's Guide*.

Examples

- Both the FrameMaker and default SGML character sets have a character code for the character ó (lowercase o with an acute accent). In FrameMaker, the character code is `0x97`; in the default

SGML character set, the character code is `0xF3`. If you want to trap the SGML character that looks like ó, you might try using this rule:

```
character map is "ó" = trap;
```

Hhowever, because you enter your read/write rules in a FrameMaker document, FrameMaker interprets that rule as:

```
character map is 0x97 = trap;
```

which is not the behavior you want. Instead, you should use this rule:

```
character map is 0xF3 = trap;
```

- By default, FrameMaker maps the SGML broken bar character to the FrameMaker solid bar character |. The rule for doing so could be written in the following equivalent ways:

```
character map is 0xA6 = "|";
character map is 0xA6 = 0x7C;
character map is "\xA6" = "\x7C";
```

- To trap the SGML broken bar character, use this rule:

```
character map is 0xA6 = trap;
```

See also

- For information on the FrameMaker character set, see the FrameMaker Character Sets guide.
- For details of the default mapping between the FrameMaker and ISO Latin-1 character sets, see Chapter 12, "Character Set Mapping."

convert referenced graphics

Use the `convert referenced graphics` rule to force the software to write out a graphic file when exporting a graphic element that uses a referenced graphic. By default, FrameMaker does not write out graphic files in this case. It is usually more advantageous to simply reference the same graphic file in both the markup and the FrameMaker document. However, you can use this rule to convert all such graphic files to a specific format.

Synopsis and contexts

```
element "gi" { . . .
writer facet "facetname" convert referenced graphics;
. . . }
```

Arguments

There are no arguments for this rule

Details

- This rule must be a subrule of a facet rule for a graphic element.

- By default, if a graphic or equation element is imported by reference, the software does not create a new graphic file for the element when exporting a FrameMaker document. You can change that behavior using this rule.

Examples

- Assume you want to convert imported graphic files in `graphic` elements which have not been edited in the FrameMaker document, to the PICT format. With the following example, the software would create PICT files for each of these graphic elements:

```
element "graphic" {
   is fm graphic element;

writer {
facet default {
   convert referenced graphics;
   export to file "$(entity).pic" as "PICT";
}}
```

- For graphic elements with a single TIFF facet, the following example converts the graphic files in the graphic element to PICT:

```
element "graphic" {
   is fm graphic element;
   writer facet "TIFF"{
   convert referenced graphics;
   export to file "$(entity).pic" as
      "PICT";
}}
```

See also

Related rules	"facet" on page 74 "export to file" on page 69 "writer" on page 168
General information on this topic	Developer Guide, page 367: Translating Graphics and Equations

do not include dtd

See "include dtd" on page 98.

do not include sgml declaration

See "include sgml declaration" on page 100.

do not output book processing instructions

See "output book processing instructions" on page 147.

drop

Use the `drop` rule to indicate information that you want discarded. Examples of information you might discard include a markup element or attribute that has no counterpart in FrameMaker, or a FrameMaker non-element marker that has no counterpart in markup.

Synopsis and contexts

```
1.attribute "attr" drop;

2.element "gi" drop;

3.element "gi" {. . .
      attribute "attr" drop;
   . . .}

4.external data entity reference drop;

5.fm attribute "attr" drop;

6.fm element "fmtag" drop;

7.fm marker type1 [, . . ., typen] drop;

8.fm variable drop;

9.processing instruction drop;
```

Arguments

attr	The name of a markup or FrameMaker attribute. Note that `fm attribute` names are case-sensitive and should appear as in the EDD. The case of SGML attribute names depends on the setting of `NAMECASE` in the SGML.dcl file—For XML attribute names are case sensitive.
gi	A markup element's name (generic identifier).
fmtag	A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.
$type_i$	A FrameMaker marker type, such as Index or Type 22.

Details

- When FrameMaker encounters something to be discarded, it makes no attempt to insert the corresponding information into the document it is creating. In the case of a dropped element, it also discards all descendant elements.
- When creating an EDD from a DTD or schema, or a DTD from an EDD, FrameMaker does not generate an element definition corresponding to a dropped element. It also removes any references to the specified element in content rules for other elements unless you've specified a `preserve fm element definition` rule for those elements.
- You can write a structure API client or XSLT stylesheet to process dropped information. Your solution must also handle retrieving discarded information if it is needed when the document is written back to its original format.
- If you use the `drop` rule in a rule, you can use no other subrules of the same rule. For example, you cannot specify that FrameMaker both drop an attribute and translate it to a FrameMaker property with the `is fm property` rule.

XSLT interaction

XSLT allows precise, context based equivalent processing to the FrameMaker `drop` rule. For consistency and maintainabilty try to avoid mixing the methods used to drop FrameMaker or XML elements.

Examples

- A markup element used instead of a processing instruction to indicate that a page or line break is desired may be discarded when the markup document is read. Text formatting rules in the EDD can be used to indicate a page break in FrameMaker; there is no need to mark the break with an element. To drop the markup element `break`, use this rule:

```
element "break" drop;
```

- By default, FrameMaker stores processing instructions that it does not recognize in non-element markers. In this way, even though FrameMaker does not perform special processing on the processing instruction, when you save the FrameMaker document back to markup, the software writes out the processing instruction so that a different application can use it. If you don't need to write out the processing instructions, you could use this rule:

```
processing instruction drop;
```

See also

Related rules

"drop content" on page 55
"unwrap" on page 162
"preserve fm element definition" on page 148

Rules mentioned in synopses	"attribute" on page 46 "element" on page 56 "external data entity reference" on page 72 "fm attribute" on page 76 "fm element" on page 77 "fm marker" on page 79 "fm variable" on page 92 "processing instruction" on page 151
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes

drop content

Use the `drop content` rule to either create a FrameMaker empty element or a markup element with no content from occurrences of *gi*.

Synopsis and contexts

```
1.element "gi" {
   is fm element "fmtag";
   reader drop content;
   }
2.element "gi" {
   is fm element "fm tag";
   writer drop content;
   }
```

Arguments

gi	A markup element's name (generic identifier).
fmtag	A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

- You can use this rule when you have an element whose content is created in a system-specific way. If you plan to rely on some system to create the content, the existing content at the time you import or export a document may not be relevant. For example, you may have a markup element intended to contain a chapter number. In FrameMaker, you use FrameMaker's formatting capabilities to have the system maintain the value. When reading in the markup document, you can drop the current content of the number element.

- Use `drop content` inside a `reader` rule when you translate markup documents to FrameMaker documents. Use it inside a `writer` rule when you translate FrameMaker documents to markup.

XSLT interaction

XSLT allows precise, context based equivalent processing to the FrameMaker `drop content` rule. For consistency and maintainabilty try to avoid mixing the methods used to drop content.

Examples

- Assume your DTD has a `toc` element that represents the table of contents for a markup document. FrameMaker can automatically generate a table of contents, which means that this markup element can have its contents dropped upon import.

```
element "toc" reader drop content;
```

- Assume the `total` element's content is computed by a structure API client. Outside the FrameMaker environment you will use a different program to perform the computation. Consequently, you do not want the value that is current when the document is exported. To discard the current value, use this rule:

```
element "total" writer drop content;
```

See also

Related rules	"drop" on page 53 "unwrap" on page 162
Rules mentioned in synopses	"element" on page 56 "reader" on page 153 "writer" on page 168
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes *Structure Import/Export API Programmer's Guide*

element

You use the `element` rule as the primary rule for translating between a markup element and its corresponding FrameMaker representation.

Synopsis and contexts

```
1.element "gi" {. . .
      subrule;
   . . .}
```

```
2.element "gi" {. . .
      transform;
   subrule;
   . . .}
```

Arguments

gi A markup element's name (generic identifier).

transform The `element` rule can include a single transform subrule is used to map to a FrameMaker object element.

`is fm colspec` specifies that the element represents a CALS table colspec. This subrule applies only to CALS tables.

`is fm cross-reference element` specifies that the element corresponds to a FrameMaker cross-reference element.

`is fm element` translates the element to a particular FrameMaker element. You use this subrule to rename the element.

`is fm equation element` specifies that the element corresponds to a FrameMaker equation element.

`is fm footnote element` specifies that the element corresponds to a FrameMaker footnote element.

`is fm graphic element` specifies that the element corresponds to a FrameMaker graphic element.

`is fm marker element` specifies that the element corresponds to a FrameMaker marker element.

`is fm span spec` specifies that the element represents a CALS table spanspec. This subrule applies only to CALS tables.

`is fm system variable element` specifies that the element corresponds to a FrameMaker system variable element.

`is fm table element` specifies that the element corresponds to a FrameMaker table element.

`is fm table part element` specifies that the element corresponds to a FrameMaker element for a particular table part, such as a table title or cell.

subrule The subrules of `element` indicate the treatment of the markup element and its attributes.

`attribute` specifies what to do with a markup element's attributes.

`drop` discards the element.

`fm attribute` specifies what to do with attributes present in the FrameMaker representation of the element but not in the markup representation of it.

`fm property` specifies what to do with FrameMaker properties associated with the element. This subrule applies only to elements that correspond to graphic, equation, table, table part, cross-reference, or marker elements.

`marker text is` specifies whether the text of a FrameMaker marker element should be element content or an attribute value in markup. This subrule applies only to marker elements.

`drop content` specifies that the content but not the structure of an element should be discarded on import of a markup document.

`end vertical straddle` indicates that the associated table cell or row element terminates a vertical table straddle. This subrule applies only to table cell or row elements.

`insert table part element` indicates that the software should insert the specified table part (title, heading or footing), even if the markup element structure does not contain the corresponding element. This subrule applies only to table elements.

`line break` determines whether to interpret line breaks in text segments in elements in the markup document as forced returns or spaces within the elements.

`start new row` indicates that the occurrence of the associated table cell element always starts a new row in the table. This subrule applies only to table cell elements.

`start vertical straddle` indicates that the associated table cell element starts a vertical table straddle. This subrule applies only to table cell elements.

`unwrap` indicates that the content of the element, but not the element itself, should be included in the translated document.

`anchored frame` tells FrameMaker what to do with graphic elements other than those with a single non-internal FrameMaker facet. This subrule applies only to graphic elements.

`drop content` specifies that the content but not the structure of an element should be discarded on export of a FrameMaker document.

`writer equation` tells FrameMaker what to do with equation elements. This subrule applies only to equation elements.

`writer facet` tells FrameMaker what to do with a graphic element that has a single non-internal FrameMaker facet. This subrule applies only to graphic elements.

`writer line break` limits the length of lines the software generates in the markup document.

`writer notation is` specifies a notation name when the element is a graphic or equation.

`writer specify size in` specifies the units of measure for the size of a graphic or equation element.

Details

If you use either the `drop` or `unwrap` subrule of an `element` rule, that subrule must be the element's only subrule. For example, you cannot both unwrap a markup element and translate it to a FrameMaker element.

Examples

- To translate the markup element `p` to the FrameMaker element `Paragraph`, use this rule:

```
element "p" is fm element "Paragraph";
```

- To translate the markup element `tab2` to a FrameMaker table element `Two Table` with two columns, use this rule:

```
element "tab2" {
   is fm table element "Two Table";
   fm property columns value is "2";
}
```

See also

Related rules	"fm element" on page 77
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes

end vertical straddle

Use the `end vertical straddle` rule inside the `element` rule for a table row or table cell to specify that the row (or the row containing the cell) indicates the end of a vertical straddle

started by some earlier table cell element. The straddle can end either before the current row or at the current row.

Synopsis and contexts

```
element "gi" {
   is fm table row_or_cell element ["fmtag"];
   reader end vertical straddle "name1" [, . . . "namen"]
      [before this row];
. . .}
```

Arguments

gi	A markup element's name (generic identifier).
row_or_cell	One of the keywords: `row` or `cell`.
	fmtag A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.
$name_i$	A name associated with a table straddle. Each $name_i$ must occur in a corresponding `start vertical straddle` rule.

Details

- Your DTD may contain elements that you want to format as tables in FrameMaker even though the element hierarchy does not match that required by FrameMaker for tables. In such a situation, the nature of the element hierarchy may indicate where vertical straddles begin and end. The `end vertical straddle` rule allows you to specify such elements.
- Use this rule in conjunction with the `start vertical straddle` rule. That rule specifies a table cell that indicates the first cell in a vertical straddle. In the `start vertical straddle` rule, give a name to the particular straddle started by that element. In the `end vertical straddle` rule, you must specify by name which vertical straddles started by earlier cells are ended by the occurrence of *gi*.
- If you use this rule for a table cell element, you can end only one vertical straddle. If you use it for a table row element, you can end more than one vertical straddle.
- If you use this element without the `before this row` keyword phrase, the cell or row (*gi*) specified in the rule becomes part of the straddle. If you do include that keyword phrase, then the straddle ends in the row above the one specified.

See also

Related rules	"start vertical straddle" on page 160
General information on this topic	Developer Guide, Chapter 22, Translating Tables

entity

You use the `entity` rule to translate an entity to an appropriate FrameMaker representation. With this rule, you can translate an entity to a particular character or set of characters, a reference element, a text inset, or a FrameMaker variable. If you choose to translate the entity to a text inset, you can also specify how to format that text inset in the resulting document.

Synopsis and contexts

```
1.entity "ename" {
      type_rule;
      [format_rule;]
   . . .}
2.reader entity "ename" {
      type_rule;
      [format_rule;]
   . . .}
```

Arguments

ename	An entity name.
type_rule	One of the following: `is fm char` translates the entity to a particular character in FrameMaker. `is fm reference element` translates the entity to an element whose content resides on a reference page in the FrameMaker document. `is fm text inset` translates the entity to a FrameMaker text inset. `is fm variable` translates the entity to a FrameMaker non-element variable.
format_rule	One of the following subrules can be specified, but only if *type_rule* is `is fm text inset`: `reformat as plain text` specifies that the software remove the internal structure and formatting from the text of the text inset and apply the formatting used at the insertion point. `reformat using target document catalogs` specifies that the software retain the text inset's internal structure and apply the containing document's formats and element format rules to the text. This rule is applied as if the following three options were checked when a file is imported through the File>ImportFile menu: 1. Reformat Using Target Document's catalog; 2. While importing Remove: Manual Page Breaks; and 3. While Importing Remove: Other Format Overrides. For more

information, see the section “Import text” in Chapter 9 of the Using FrameMaker guide.

`retain source document formatting` specifies that the software remove the internal structure of the text inset and retain the formatting of the text inset as it appeared in the source document.

Details

- If you use the `entity` rule at the highest level, then it applies both on import and export. If you use it inside a `reader` rule, then FrameMaker translates the entity as specified when importing a markup document, but does not create an entity reference on export.
- For SGML, while you can use this rule to translate any entity type to a text inset, we recommend you convert only `SDATA` entities to text insets. Note that the source file for such a text inset must be a format FrameMaker can automatically filter. Also, such a text inset cannot use a markup document as the source file.
- For XML and SGML, FrameMaker imports external text entities as text insets by default. The source files for these insets can be markup or text files. The software stores entity information on the Entity Declarations reference page so it can export the text inset as an external text entity.
- For XML, `SDATA` and `CDATA` entities are not allowed.

Examples

- To translate the text entity `mn` to the FrameMaker variable `Manual Name`, use this rule:

  ```
  entity "mn" is fm variable "Manual Name";
  ```

  Suppose the text entity mn is declared as `<!ENTITY mn "Developer’s Guide">`, and the template for the application does not contain a variable named `Manual Name`. Then the software will create a FrameMaker variable named `Manual Name` defined as `Developer’s Guide` and replace the reference in the text with the variable text `Developer’s Guide`.

  However, if a FrameMaker variable named `Manual Name`, defined for example as `My Favorite Manual`, currently exists in the template for the application, when importing SGML, the software will not create a new variable nor modify the existing one. It will replace the reference in the text with the variable text `My Favorite Manual`. When importing XML, it does modify the variable definition.

- To have FrameMaker create a text inset for the `legalese` entity using the text in the file `legal.fm` and to have the software format that text inset as it appears in `legal.doc`, use this rule:

  ```
  entity "legalese" {
     is fm text inset "legal.fm";
     retain source document formatting;
  }
  ```

See also

General information on this topic	Developer Guide, Chapter 21, Translating Entities and Processing Instructions
	Developer Guide, Chapter 25, Translating Variables and System Variable Elements

entity name is

Use the `entity name is` rule only in an `element` rule for a graphic or equation element to provide information the software needs when writing a document containing graphics or equations to markup. The `entity name is` rule determines the name FrameMaker gives an entity reference it generates for the graphic or equation.

Synopsis and contexts

```
1.element "gi" {
      is fm equation element ["fmtag"];
      writer equation entity name is "ename";
   . . .}}
2.element "gi" {
      is fm graphic element ["fmtag"];
      writer anchored frame entity name is "ename";
   . . .}}
3.element "gi" {
      is fm graphic element ["fmtag"];
      writer facet "facetname" entity name is "ename";
   . . .}}
```

Arguments

gi

A markup element's name (generic identifier).

fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

facetname

A facet name. The string for the facetname must exactly match the string for the facetname in the FrameMaker document. To determine a graphic file's facetname, select the graphic, click Graphics>ObjectProperties, and observe the facetname in the dialog box.

ename

A string representing the base name for an entity name.

Details

By default, when FrameMaker exports an external data entity for a graphic or equation, it uses the entity name that is stored with the graphic inset. If there is no such entity name, the software generates a name for the entity based on the element name. You use the `entity name is` rule to change this behavior.

The entity name you specify is a base name FrameMaker uses to generate a unique entity name. When it needs to create a new entity name, FrameMaker adds an integer to the name specified by *`ename`* to create a unique name.

If the keyword `facet` is used, the rule applies to a graphic element that contains only a single facet with the name specified by *`facetname`*. This occurs when the graphic element is an anchored frame containing only a single imported graphic object whose original file was in the *`facetname`* graphic format. You can use this rule multiple times if you want FrameMaker to treat several file formats differently.

Examples

- Assume you have a markup element `graphic` that corresponds to graphic elements in FrameMaker. Suppose further that some of the graphic elements in FrameMaker contain imported-by-copy graphics, or contain modifications to a graphic inset using FrameMaker graphic tools, or contain just graphic objects drawn using FrameMaker graphic tools. On export, the software must create new graphic files for these elements and declare entities for them. By default, FrameMaker would declare entities for these graphic elements based on the element name "graphic," for example, `graphic1`, `graphic2`, and so on. To specify that the names of the entities associated with such successive graphic elements have the form `car1`, `car2`, and so on, use this rule:

  ```
  element "graphic" {
     is fm graphic element;
     writer anchored frame entity name is "car";
  }
  ```

- Assume with a single facet graphics in the `car` element sometimes use the IGES file format and sometimes use the TIFF file format. Also assume that the DTD for the application does not currently contain entity declarations for the imported-by-reference graphic files. By default, the software would declare entities for all such graphics based on the element name "car," for example, `car1`, `car2`, and so on. If you want to name the entities for the IGES graphics `icar` and the entities for the TIFF graphics `tcar`, then use this rule:

  ```
  element "car" {
     is fm graphic element;
     writer facet "IGES" entity name is "icar";
     writer facet "TIFF" entity name is "tcar";
  }
  ```

See also

Related rules	"export to file" on page 69 "notation is" on page 145 "specify size in" on page 157
Rules mentioned in synopses	"element" on page 56 "is fm graphic element" on page 114 "is fm equation element" on page 111 "anchored frame" on page 43 "equation" on page 65 "facet" on page 74 "writer" on page 168
General information on this topic	Developer Guide, Chapter 23, Translating Graphics and Equations

equation

Use the `equation` rule only in an `element` rule for an equation element, to provide information the software needs when writing to markup a document containing equations. FrameMaker creates graphic files to represent equations. Use this rule to specify information about the files FrameMaker creates for instances of the equation element. By default, the software creates a file in CGM format for each equation, and the filename is based on the element name. Also, by default, if the equation element is associated with an external data entity, then the entity name is based on the element name.

Synopsis and contexts

```
element "gi" {
   is fm equation element ["fmtag"];
   writer equation subrule;
. . .}
```

Arguments

gi	A markup element's name (generic identifier). *fmtag* A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.
subrule	An `equation` rule can have the following subrules:

`entity name is` tells the software how to create the base name for the entity associated with this element type.

`export to file` tells the software to write a new file for the associated external data entity.

`notation is` specifies the data content notation of the entity file.

`specify size in` specifies the units to use when writing the file.

Examples

Assume you have an element named `math` with an attribute of type `Entity` that is mapped to the `fm property` entity for this element. If you want to create TIFF files for the equations in a document named `mytest.doc`, you might use this rule:

```
element "math" {
   is fm equation element;
   writer equation export to file "$(docname).eqn" as "TIFF";
}
```

The software will create graphic files for each equation in `mytest.doc` named `mytest1`, `mytest2`,...and will declare entities named `math1`, `math2`, ...for each graphic.

See also

Related rules	"anchored frame" on page 43 "facet" on page 74 "is fm graphic element" on page 114
Rules mentioned in synopses	"element" on page 56 "is fm equation element" on page 111 "writer" on page 168
General information on this topic	Developer Guide, Chapter 23, Translating Graphics and Equations

export dpi is

You use the `export dpi` rule only in an `element` rule for a graphic or equation element, to provide information the software needs when writing a document containing graphics or

equations to markup. The `export dpi` rule tells FrameMaker the dpi setting to use for an exported graphic file.

Synopsis and contexts

```
1.element "gi" {
      is fm equation element ["fmtag"];
      writer equation
         export dpi is number;
      . . .
   . . .}

2.element "gi" {
      is fm graphic element ["fmtag"];
      writer anchored frame
         export dpi is number;
      . . .
   . . .}

3.element "gi" {
      is fm graphic element ["fmtag"];
      writer facet "facetname"
         export dpi is number;
      . . .
   . . .}
```

Arguments

gi — A markup element's name (generic identifier).

fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

facetname

A facet name. The string for the facetname must exactly match the string for the facetname in the FrameMaker document. To determine a graphic file's facetname, select the graphic, click Graphics>ObjectProperties, and observe the facetname in the dialog box.

number — The required dpi value.

Details

- In the absence of this rule, FrameMaker uses the dpi setting associated with the graphic file. If there is no setting associated with the graphic, the software assumes a value of 300.

- In Windows, if the initialization file for a graphics filter specifies a dpi setting that setting overrides this rule whenever that filter is used to export a graphic file.
- If the keyword `facet` is used, the rule applies to a graphic element that contains only a single facet with the name specified by *`facetname`*. This occurs when the graphic element is an anchored frame containing only a single imported graphic object whose original file was in the *`facetname`* graphic format. You can use this rule multiple times if you want FrameMaker to treat several file formats differently.

Examples

- Assume you export the FrameMaker file `Math.doc` and have the following rule:

```
element "eqn" {
   is fm equation element "Equation";
   writer equation
   export dpi is 72;
}
```

  When FrameMaker finds an instance of the `Equation` element, it exports equations as graphic files at 72 dpi.

- Assume you have the rule:

```
element "imp" {
   is fm graphic element;
   writer facet "TIFF"{
   convert referenced graphics;
   export dpi is 1200;
   export to file "$(entity).tif";
}}
```

  This rule tells FrameMaker for every graphic element with a single TIFF facet, it should write a new graphic file with a dpi of 1200, using the entity name as part of the graphic file's filename.

See also

Related rules

"convert referenced graphics" on page 51
"entity name is" on page 63
"notation is" on page 145
"specify size in" on page 157

Rules mentioned in synopses	"element" on page 56 "is fm graphic element" on page 114 "is fm equation element" on page 111 "anchored frame" on page 43 "equation" on page 65 "facet" on page 74 "writer" on page 168
General information on this topic	Developer Guide, Chapter 23, Translating Graphics and Equations

export to file

You use the `export to file` rule only in an `element` rule for a graphic or equation element, to provide information the software needs when writing a document containing graphics or equations to markup. The `export to file` rule tells FrameMaker how to write the file name when it creates a new graphic file, and optionally the graphic format for the file.

Synopsis and contexts

```
1.element "gi" {
      is fm equation element ["fmtag"];
      writer equation
         export to file "fname" [as "format"];
   . . .}

2.element "gi" {
      is fm graphic element ["fmtag"];
      writer anchored frame
         export to file "fname" [as "format"];
   . . .}

3.element "gi" {
      is fm graphic element ["fmtag"];
      writer facet "facetname"
         export to file "fname" [as "format"];
   . . .}
```

Arguments

gi	A markup element's name (generic identifier).
	fmtag

	A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.
facetname	A facet name. The string for the facetname must exactly match the string for the facetname in the FrameMaker document. To determine a graphic file's facetname, select the graphic, click Graphics>ObjectProperties, and observe the facetname in the dialog box.
fname	A base filename which can be either absolute or relative to the output directory. Note: If path information is included in fname, use double backslashes to translate path backslashes correctly. The *fname* argument can contain the variables `$(docname)` and `$(entity)`, described below.
format	A file data content format code, such as TIFF or PICT. See Developer Guide, Chapter 23, Translating Graphics and Equations for a complete list of graphic format codes. `format` must be one of these code names.

Details

- By default, if a graphic element has a single facet (other than a FrameMaker internal facet) that was imported by reference, FrameMaker does not create a new graphic file. On export, the original file will be associated with a markup graphic element via the `file` attribute, or via the `entity` attribute plus a corresponding entity declaration. You can use the `convert referenced graphics` rule to force FrameMaker to export such graphic files.
- If the keyword `facet` is used, the rule applies to a graphic element that contains only a single facet with the name specified by *facetname*. This occurs when the graphic element is an anchored frame containing only a single imported graphic object whose original file was in the *facetname* graphic format. In this case, the rule is only executed if the `convert referenced graphics` rule is also used. Otherwise, it is ignored.You can use this rule multiple times if you want FrameMaker to treat several file formats differently.
- If your rules specify the software will write a graphic file, if a graphic element has a single facet (other than a FrameMaker internal facet), FrameMaker writes the graphic file in that format by default. It writes the graphic file for equation elements and all other graphic elements in CGM format by default.

  If you supply a *format* argument, you must first make sure that the format is one known to FrameMaker. For information on which graphic export filters the software provides and on how to add new ones, see Developer Guide, Chapter 23, Translating Graphics and Equations.
- The *fname* argument can use these variables:

Variable	Meaning
`$(entity)`	The value of the corresponding markup element's `entity` attribute. If the source of the graphic inset wasn't originally an entity, this variable defaults to a unique name based on the name of the element. You can change this name using the `entity name is` rule.

Variable	Meaning
`$(docname)`	The name of the FrameMaker file, excluding any extension or directory information.

- The *`fname`* argument is used as a template for the actual filename FrameMaker generates for a particular graphic or equation element. FrameMaker takes the filename specified with the *`fname`* argument and may append an integer to the filename to ensure uniqueness of the filename. For an example of this behavior, see the first example below.

Examples

- Assume you export the FrameMaker file `Math.fm` and have the following rule:

```
element "eqn" {
   is fm equation element "Equation";
   writer equation
      export to file "$(docname).eqn" as "PICT";
}
```

When FrameMaker finds an instance of the `Equation` element, it generates filenames of the form `MathN.eqn` until it finds a name that does not collide with an already existing file. For example, if you already have files in the specified directory named `Math1.eqn` and `Math2.eqn`, the software writes the first equation to a file named `Math3.eqn`. FrameMaker writes the equation file in PICT format, instead of the default CGM format.

- Assume you have the rule:

```
element "imp" {
   is fm graphic element;
   writer facet "TIFF" {
convert referenced graphics;
export to file "$(entity).tif";
}}
```

This rule tells FrameMaker that if it encounters a graphic element with an imported graphic file with a single TIFF facet, it should write that graphic to the file specified by `$(entity).tif`.

See also

Related rules

"convert referenced graphics" on page 51
"entity name is" on page 63
"notation is" on page 145
"specify size in" on page 157

Rules mentioned in synopses	"element" on page 56 "is fm graphic element" on page 114 "is fm equation element" on page 111 "anchored frame" on page 43 "equation" on page 65 "facet" on page 74 "writer" on page 168
General information on this topic	Developer Guide, Chapter 23, Translating Graphics and Equations

external data entity reference

Use the `external data entity reference` rule to drop references to all external data entities. By default, FrameMaker stores such references as the marker text in non-element Entity Reference markers.

Synopsis and contexts

```
external data entity reference drop;
```

ArgumentsNone.

Details

- In markup, the values of general entity name attributes, such as those used with graphics, are not considered entity references. This rule does not affect how FrameMaker treats general entity name attributes. In XML such entity name attributes are the only way to reference non-parsed entities such as graphics.
- When you translate a markup document to FrameMaker, when the software encounters an external data entity reference such as:

  ```
  &door;
  ```

  it stores the reference as the text of a non-element DOC Entity Reference marker by default, with the following marker text:

  ```
  door
  ```

  When you translate a FrameMaker document to markup, it outputs the marker text of non-element DOC Entity Reference markers as entity references.

Examples

To discard all external data entity references, use this rule:

```
external data entity reference drop;
```

See also

Rules mentioned in synopses	"drop" on page 53
General information on this topic	Developer Guide, Chapter 21, Translating Entities and Processing Instructions

external dtd

Use this rule to specify how an exported markup instance refers to the current structure application's DTD. By default, FrameMaker uses the name of the file containing the DTD as the system identifier in the external identifier for the DTD. The `external dtd` rule provides the software with a different external identifier. The different forms of the rule allow specification of a system identifier, public identifier, or both.

Synopsis and contexts

```
1.writer external dtd is system;
2.writer external dtd is system "sysid";
3.writer external dtd is public "pubid";
4.writer external dtd is public "pubid" "sysid";
```

Arguments

sysid	A system identifier.
pubid	A public identifier.

Details

- Use this rule when you export FrameMaker documents to markup documents To use this rule, you must have a DTD specified for the current structure application in the `structapps.fm` file.
- By default, FrameMaker does not reproduce the DTD in the document type declaration subset. Instead, it uses the filename of the DTD that was specified in the structure application to write a document type declaration of the form:

  ```
  <!DOCTYPE doctype SYSTEM "fname" [. . .
  ```

  where *doctype* is the document type name and *fname* is the DTD filename specified in the structure application. This rule allows you to specify different system and public identifiers.
- To output both external DTD and Schema with an XML document, use this rule and specify a Schema file for output in the XML structure application (in `structapps.fm`). This rule modifies how the external DTD is written.

To output Schema only, with no DTD, specify only the Schema file, not the DTD, in `structapps.fm`. You do not need to use this rule.

- You cannot use the `external dtd` rule in the same read/write rules file as the `include dtd` rule.

Examples

- To specify a local DTD as an external DTD and include the path with the filename, you could use this rule:

  writer

  ```
  external dtd is
     system "/doc/dtds/manuals.dtd";
  ```

  Note that the Windows platform requires two backslashes in paths in the rules file in order to translate as one backslash.

- To specify and locate the CALS DTD as an external DTD, you could use this rule:

  ```
  writer external dtd is
     public "-//USA-DOD//DTD MIL-M-38784B//EN"
     "/doc/dtds/cals.dtd";
  ```

- To specify just the CALS DTD as an external DTD using a public identifier, you could use this rule:

  ```
  writer external dtd is
  public "-//USA-DOD//DTD MIL-M-38784B//EN";
  ```

  You could then specify the location of the DTD in the structure application using the `EntitiesLocation` element. A DTD is an entity in the strictest sense.

See also

Related rules	"include dtd" on page 98
	"include sgml declaration" on page 100
	"write structured document" on page 166
	"write structured document instance only" on page 167
Rules mentioned in synopses	"writer" on page 168

facet

Use the `facet` rule only in an `element` rule for a graphic element, to provide information the software needs when writing a document containing graphics to markup. The `facet` rule applies only when a graphic element is an anchored frame containing only a single imported graphic

object whose original file was in the *facetname* graphic format. Use this rule to specify information about the graphic file and/or entity declaration for instances of the graphic element.

Synopsis and contexts

```
element "gi" {
   is fm graphic element ["fmtag"];
   writer facet "facetname" subrule;
. . .}
```

Arguments

gi — A markup element's name (generic identifier).

fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

facetname — The name of the particular facet to which this rule applies, or "`default`" for all facets.

subrule — A `facet` rule can have the following subrules:

`convert referenced graphics` tells the software to create new graphic files for imported graphic files with a single facet.

`entity name is` tells the software how to create the base name for the entity associated with this element type.

`export to file` tells the software the name to use for graphics it creates, and optionally, the graphic format to which it should convert.

`notation is` specifies the data content notation of the entity.

`specify size in` specifies the units to use when writing the file.

Details

To specify all facets, use the keyword default for the facetname argument. For example:

```
element "pict" {
is fm graphic element "Picture";
writer {
   facet default {
   convert referenced graphics;
   export to file "$(entity).tif" as "TIFF";
   . . .
}}}
```

will convert every imported graphic file in the document to a TIFF file, no matter what its original facet was.

Examples

By default, FrameMaker does not create a new graphic file for a graphic element that originated as an external entity, and was not modified by the user in any way. Assume you want the software to generate a graphic file for every imported TIFF file, whether it was modified or not. Then you could use this rule:

```
element "pict" {
   is fm graphic element "Picture";
   writer {
      facet "TIFF" {
      convert referenced graphics;
      export to file "$(entity).tif" as "TIFF";
}}}
```

See also

Related rules	"anchored frame" on page 43 "convert referenced graphics" on page 51 "equation" on page 65
Rules mentioned in synopses	"element" on page 56 "is fm equation element" on page 111 "is fm graphic element" on page 114 "writer" on page 168
General information on this topic	Developer Guide, Chapter 23, Translating Graphics and Equations

fm attribute

You use the `fm attribute` rule with the "drop" subrule to discard an attribute that you've defined for a FrameMaker element but that does not exist on the corresponding markup element. Read/write rules do not support double-byte characters, so you cannot use this rule to drop attributes with double-byte characters in their names.

Synopsis and contexts

```
1.fm attribute "attr" drop;

2.element "gi" {. . .
      fm attribute "attr" drop;
   . . .}
```

Arguments

attr	A FrameMaker attribute name.

gi A markup element's name (generic identifier).

Examples

- Assume the element `chapter` exists in both the markup and FrameMaker representations of your documents. In FrameMaker, you use the `XRefLabel` attribute in formatting cross-references to this element. Since this attribute exists only for formatting purposes, you don't want it in the markup document. To drop this attribute on export, use this rule:

```
element "chapter" {
   is fm element;
   fm attribute "XRefLabel" drop;
}
```

- If you use the `XRefLabel` attribute on many elements for the same purpose, you can discard it from all elements on export with this rule:

```
fm attribute "XRefLabel" drop;
```

- If you want to keep the `XRefLabel` attribute on the `appendix` element, but drop it from all others, use these rules:

```
element "appendix" {
   is fm element;
   attribute "xreflab" is fm attribute "XRefLabel";
}
fm attribute "XRefLabel" drop;
```

  Note that the order of these rules is not important. If you reversed them, the `XRefLabel` attribute would still be correctly interpreted for the `appendix` element, since that reference to the attribute is more specific. Note also that case is sensitive for fm attribute names.

See also

Related rules	"attribute" on page 46 "is fm attribute" on page 104
Rules mentioned in synopses	"element" on page 56 "drop" on page 53
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes

fm element

Use the `fm element` rule to tell FrameMaker what to do on export with FrameMaker elements that do not correspond to markup elements. Read/write rules do not support double-byte

characters, so you cannot use this rule to process elements with double-byte characters in their names.

Synopsis and contexts

```
1.fm element "fmtag" drop;
2.fm element "fmtag" unwrap;
```

Arguments

fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

- Use this rule when you export FrameMaker documents to markup documents.
- If you use this rule, you may want to write a structure API client to handle the export of the element or to create it on import.
- The first version of this rule discards the FrameMaker element on export. The second version inserts the contents of *fmtag* in the corresponding markup document, but not *fmtag* itself.
- If you use this rule to unwrap FrameMaker cross-reference elements or system variable elements, those elements become text in the resulting markup document.

XSLT interaction

XSLT allows precise, context based equivalent processing to the FrameMaker `drop` and `unwrap` rules. For consistency and maintainabilty try to avoid mixing the methods used to drop or unwrap FrameMaker elements.

Examples

- If `Chapter Number` is a FrameMaker element that you want to discard on export, use this rule:

  ```
  fm element "Chapter Number" drop;
  ```

  If you use this rule and want to create this element on import, you need to write a structure API client.

- If `Modification Date` is a FrameMaker system variable element that you wish to translate to text on export to markup, use this rule:

  ```
  fm element "Modification Date" unwrap;
  ```

See also

Related rules "element" on page 56
"is fm element" on page 110

Rules mentioned in synopses	"drop" on page 53 "unwrap" on page 162
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes

fm marker

On export, you use the `fm marker` rule to tell FrameMaker what to do with non-element markers other than markers of the type reserved for storing processing instructions, PI entities, and external data entities. (By default, Structure PI markers are reserved for processing instructions, and Entity Reference markers are reserved for external data entities.) In the absence of a rule to the contrary, the software creates processing instructions for non-element markers. You can also choose to discard them. Read/write rules do not support double-byte characters, so you cannot use this rule to process markers with double-byte characters in their names.

Synopsis and contexts

```
fm marker ["type1", . . ., "typen"] drop;
fm marker ["type1", . . ., "typen"] is processing instruction;
```

Arguments

$type_i$	The name of a FrameMaker marker type.

Details

- If $type_i$ is specified, this rule applies only to markers of that type.

  If no $type_i$ is specified, this rule applies to all non-element markers other than markers of the reserved type. For information on what the software does with the reserved marker type, see Developer Guide, Chapter 26, Translating Markers

- You can have multiple occurrences of this rule in a rules file, to determine different treatment for different FrameMaker markers. You can have only one occurrence of the rule with no explicitly listed markers. A given marker type can be explicitly mentioned in only one occurrence of this rule.

- The order of `fm marker` rules is not important. A more specific occurrence of the rule always takes precedence over a more general occurrence. For example, the following rules:

  ```
  fm marker "Index" is processing instruction;
  fm marker drop;
  ```

  have the same effect, exporting only index markers as processing instructions, if they occur in this order:

  ```
  fm marker drop;
  fm marker "Index" is processing instruction;
  ```

XSLT interaction

XSLT allows precise, context based equivalent processing to the FrameMaker `drop` rule. For consistency and maintainabilty try to avoid mixing the methods used to drop FrameMaker non-element markers.

Examples

- To discard all non-element markers, use this rule:

```
fm marker drop;
```

- To discard non-element conditional text markers but retain all others as processing instructions, use this rule:

```
fm marker "Conditional Text" drop;
```

- To retain only Index and Hypertext markers as processing instructions and drop all other non-element markers, use the following set of rules:

```
fm marker "Index", "Hypertext" is processing instruction;
fm marker drop;
```

See also

Related rules	"is fm marker element" on page 115
Rules mentioned in synopses	"drop" on page 53 "is processing instruction" on page 141
General information on this topic	Developer Guide, Chapter 26, Translating Markers

fm property

You use the `fm property` rule to determine values for properties defined for certain types of FrameMaker constructs that you do not want to represent as markup attributes.

Synopsis and contexts

1.Cross-reference elements

```
element "gi" {
   is fm cross-reference element ["fmtag"];
   fm property cross-reference format value is val;
. . .}
```

1.1 Arguments

val	A valid cross-reference format name. These names are case-sensitive and must appear in the rule the same as in the structure application's template.

2.Graphic elements or equation elements

```
element "gi" {
   is fm graphic_or_equation element ["fmtag"];
   fm property prop value is "val";
. . .}
```

MathML equations

```
element "mathml" {
 is fm mathmlequation element "RuleML";
attribute "sideways" is fm property composedpi;
. . .}
```

2.1 Arguments

prop

- `alignment` Indicates the anchored frame's horizontal alignment on the page.

  val

  `–aleft` Align left

  `–acenter` Align center

  `–aright` Align right

  `–ainside` Align inside, or closest to the binding margin.

  `–aoutside` Align ouside, or farthest from the binding margin.

- `angle` Indicates an angle of rotation for the anchored frame that contains the graphic. You must specify exact multiples of 90 degrees. Otherwise, the value is ignored and the graphic is imported at 0 degrees (default).

  val examples:

  `–0` No rotation (default)

  `–90` Rotate 90 degrees clockwise

  `––90` Rotate 90 degrees anticlockwise

  `–180` Rotate 180 degrees

  `–270` Rotate 270 degrees.

- `baseline offset` Indicates how far from the baseline of a paragraph to place an anchored frame. Baseline offset is relevant only for anchored frames whose position attribute is one of inline, sleft, sright, snear, or sfar.

  val A number plus a valid unit of measure, e.g. "12pt", "10mm". If not supplied, the value is 0.

- `cropped` Indicates whether a wide graphic should be allowed to extend past the margins of the text frame. The `cropped` property is

relevant only for anchored frames whose `position` attribute is one of `top`, `below`, or `bottom`.

val

`–0` The graphic may extend past the margins of the text frame.

`–1` (Default) The graphic is cropped at the margins of the text frame.

- `dpi` Indicates how to scale an imported graphic object.

  val The value of the dpi attribute must be an integer greater than 0. If not supplied, the default value is 72.

- `entity` Provides the entity name of the imported graphic. This rule limits the graphic import to a single, fixed file for all instances of the element.

  val A valid entity name as defined in an entity declaration in the markup instance.

- `file` Provides the file name of the imported graphic. This rule limits the graphic import to a single, fixed file for the element.

  val A valid file name for an imported graphic.

- `floating` Indicates whether the graphic should be allowed to float from the paragraph to which it is attached. The `floating` property is relevant only for anchored frames whose `position` property is one of `top`, `below`, or `bottom`.

  val

  `–0` (Default) No float, the graphic must stay with the paragraph.

  `–1` Allow float.

- `height` Indicates the height of the anchored frame.

  val The value for a single imported graphic object is the sum of the height of the object plus twice the value of the vertical offset property.

- `horizontal offset` Indicates how far the graphic object is offset from the right and left edges of the anchored frame.

  val A number with a valid unit of measure. If not supplied, the delault value is 6.0pt.

- `import angle` Indicates an angle of rotation in degrees for the graphic inside its anchored frame.

  val A real number, if not supplied, the default value is 0.0.

- `import by reference or copy` Indicates whether an imported graphic object remains in a separate file or is copied into the FrameMaker document on import from markup.

  val

`-ref` (Default) The object is referenced and will not be copied into the document.

`-copy` The object will be copied into the document.

- `import size` indicates the size of the imported graphic object by specifying a width and height.

  `val` Two numbers, separated by a space, with a valid units of measure. The first measurement is the width and the second is the height. If no unit of measure is supplied, points are assumed. Example: "100mm 50mm".

- `near-side offset` Indicates how far to set a frame from the text frame to which the frame is anchored. It is relevant only for anchored frames whose position attribute is one of `sleft`, `sright`, `snear`, or `sfar`.

  `val` A number plus a valid unit of measure, e.g. "12pt", "10mm". If not supplied, the value is 0.

- `position` Indicates where on the page to put the anchored frame. If not supplied, the value is below.

  `val` Possible anchoring position values are as follows:

  `-inline` At insertion point.

  `-top` At top of column.

  `-below` Below current line.

  `-bottom` At bottom of column.

  `-sleft` Outside column - left side.

  `-sright` Outside column - right side.

  `-snear` Outside column - right side.

  `-sfar` Outside column - side closer to the page edge.

  `-sinside` Outside column - side closer to the binding.

  `-soutside` Outside column - side farther from the binding.

  `-tleft` Outside text frame - left side.

  `-tright` Outside text frame - right side.

  `-tnear` Outside text frame - side closer to the page edge.

  `-tfar` Outside text frame - side farther from the page edge.

  `-tinside` Outside text frame - side closer to the binding.

  `-toutside` Outside text frame - side closer to the binding.

  `-runin` Run into paragraph.

- `sideways` Indicates that the imported graphic will be flipped left to right to give a mirror image.

val

–0 (Default) No flip.

–1 Flip left/right.

- vertical offset Indicates how far the graphic object is offset from the top and bottom edges of the anchored frame.

  val A number plus a valid unit of measure. If not supplied, the delault value is 6.0pt.

- width Indicates the width of the anchored frame.

  val The value for a single imported graphic object is the sum of the width of the object plus twice the value of the horizontal offset property.

- poster The name of the file displayed as the poster for an imported media file. For SWF files, FrameMaker displays the first frame of the SWF file as the poster. For a SWF file whose first frame cannot be read, and for all other media types, FrameMaker displays the relevant placeholder image.

  val A valid path to the location of the poster file.

- graphic name A name assigned to the imported object, for easy identification when linking to it.

  val A string representing the graphic name.

- activate in PDF A boolean value indicating whether or not the graphic element is activated when the PDF file containing it, is opened. The default value is False.

  val

  –0 (Default) Not activated in PDF.

  –1 Activated in PDF.

- open in pop-up window A boolean value indicating whether or not the graphic element in a PDF file is displayed in a new frame, when clicked.

  val

  –0 (Default) Not opened in pop-up window.

  –1 Opened in pop-up window.

- javascript file The JavaScript file that is attached to the graphic object with a U3D facet.

  val A valid path to the location of the JavaScript file.

- U3D view The object perspectives available for a 3D object. The selected view is rendered when the document is saved. All predefined

views of the 3D object are available when the document is converted to a PDF file. The last view that you selected in the document, before saving, becomes the default view in the PDF.

val A valid object perspective available for the 3D object.

- `background color` The color of the background for the anchored frame containing the 3D file.

  val A valid color for the background.

- `render mode` The rendering mode for an imported 3D object. The default value is Solid.

  val A valid rendering mode.

- `lighting` The lighting scheme to cast a 3D object using different light sources. The default lighting scheme for all 3D objects is Lights From File.

  val A valid lighting scheme for casting the 3D object.

- `link to text` A 3D object and a destination marker that links the object to text in the document.

  val Number of links from the 3D object and link name - destination marker pairs.

  For example, `linktotext="2;Ground_Plane=newlink aa;Blue_Sphere=newlink cc;"`

- `compose Dpi` The resolution, in Dpi, of the image composed by the MathML editor for a MathML object, displayed in FrameMaker.

  val A valid resolution for the MathML object.

- `alt text` The text that is displayed when a graphic element cannot be rendered.

  val A string for the alternate text.

- `font size` The size of the font used for MathML objects.

  val A valid size for the MathML object fonts.

3.Marker elements

```
element "gi" {
   is fm marker element ["fmtag"];
   fm property prop value is val;.
. . .}
```

3.1 Arguments

prop

- `marker text` Provides a fixed text string for all instances of the marker.

val Any valid marker text string.

- `marker type` Identifies the type of marker if not provided by a markup attribute.

  val A valid marker type name.

4.Table elements

```
element "gi" {
   is fm table element ["fmtag"];
   fm property prop value is val;.
. . .}
```

4.1 Arguments

prop

- `column ruling` Specifies whether all columns should have ruling on their right side. This property does not specify the style or weight of the ruling. The default ruling is defined by the relevant table format in the structured template.

  val

  –0 Columns have no ruling.

  –1 Columns have ruling.

- `column widths` The width of successive columns in the table. On import from markup these widths are reapplied regardless of any changes made by the user.

  val Each value is either an absolute width or a width proportional to the size of the entire table. If proportional widths are used, the `pgwide` attribute or `page wide` property determines the table overall width. Example for a three column table:
  "22mm 40mm 100mm".

- `columns` The number of columns in the table. This is essential to the correct rendering of the table if the markup does not state the number of columns as an attribute value.

  val An integer greater than 0.

- `page wide` This is relevant only to tables whose columns use proportional widths on pages with more than a single column. In this case, the attribute indicates whether the entire table should be the width of the column in which it is anchored, or the width of the overall text frame.

  val

  –0 (Default) The table is the width of the text column.

  –1 The table is the width of the text frame.

- `row ruling` Specifies whether all rows should have ruling on their bottom side. This property does not specify the style or weight of the ruling. The default ruling is defined by the relevant table format in the structured template.

  val

  –0 Rows have no ruling.

  –1 Rows have ruling.

- `table border ruling` Specifies whether the table should have ruling around its outside borders. This property does not specify the style or weight of the ruling. The default ruling is defined by the relevant table format in the structured template.

  val

  –all Rows have no ruling.

  –top Rows have ruling.

- `table format` Specifies the table format for all instances of the FrameMaker table element.

  val A name of a table format that is present in the application's structured template.

5.Table cell elements

```
element "gi" {
   is fm table cell element ["fmtag"];
   fm property prop value is val;.
. . .}
```

5.1 Arguments

prop

- `column name` Associates a name with a cell in a given column.

  val A valid column name as defined in a colspec.

- `column number` Indicates the column number that the cell will start in. This rule is used when the column number is not available in the markup and requires each cell in a given row to have a unique element name.

  val An integer greater than 0.

- `column ruling` Specifies whether the cell should have ruling on its right side. This property does not specify the style or weight of the ruling. The default ruling is defined by the relevant table format in the structured template.

  val

  –0 Cell has no right side ruling.

–1 Cell has right side ruling.

- `end column name` Specifies the name of a column that ends a straddle.

  val A valid column name as defined in a colspec.

- `horizontal straddle` Specifies the number of columns a straddled cell spans.

  val An integer greater than 1 and no greater than the number of columns.

- `more rows` Specifies the number of additional rows a straddled cell spans.

  val An integer greater than 1 and no greater than the number of rows in the table part. The total number of rows the cell occupies is `more rows`+1.

- `rotate` Indicates how much to rotate the contents of a cell.

  val The CALS model restricts this property to a boolean value, where 1 indicates a rotation of 90 degrees anti-clockwise. FrameMaker extends the possible values to allow rotations of 0, 90, 180, and 270 degrees.

- `row ruling` Specifies whether the cell should have ruling on its bottom side. This property does not specify the style or weight of the ruling. The default ruling is defined by the relevant table format in the structured template.

  val

  –0 Cell has no bottom side ruling.

  –1 Cell has bottom side ruling.

- `span name` Applies a predefined CALS spanspec, starting at this cell.

  val A valid `spanspec` name.

- `start column name` Specifies the name of a column that begins a horizontal straddle.

  val A valid column name as defined in a colspec.

- `vertical straddle` Specifies the number of rows a straddled cell spans.

  val An integer greater than 1 and no greater than the number of rows in the section (head, body or foot) of the table that contains the starting cell.

- `cell angle` Specifies the angle of rotation

  val The degrees.

- `use fill override` Specifies whether a custom fill percentage overrides the fill percentage specified in the table format.

  val

  –0 Cell has no fill override.

  –1 Cell has fill override.

- `fill override` Specifies the fill percentage for the cell.

  val A valid fill percentage.

6.Table row elements

```
element "gi" {
   is fm table row element ["fmtag"];
   fm property prop value is val;.
. . .}
```

6.1 Arguments

prop

- `maximum height` Specifies the maximum height for each row in the table.

  val A number plus a valid unit of measure, e.g. “24pt”, “15mm”. If not supplied, the maximum height of the row is not limited.

- `minimum height` Specifies the minimum height for each row in the table.

  val A number plus a valid unit of measure, e.g. “12pt”, “9mm”. If not supplied, the minimum height of the row is not limited.

- `row type` Sets the row type.

  val

  –heading

  –body

  –footing

- `row ruling` Specifies whether the cell should have ruling on its bottom side. This property does not specify the style or weight of the ruling. The default ruling is defined by the relevant table format in the structured template.

  val

  –0 Cell has no bottom side ruling.

  –1 Cell has bottom side ruling.

7.For CALS table colspecs:

```
element "gi" {
   is fm colspec;
   fm property prop value is val;
. . .}
```

7.1 Arguments

prop

- cell alignment character
- cell alignment offset
- cell alignment type
- column name
- column number
- column ruling
- column width
- row ruling
- vertical alignment

```
8.element "gi" {
      is fm spanspec;
      fm property prop value is val;
   . . .}
```

8.1 Arguments

prop

- cell alignment character
- cell alignment offset
- cell alignment type
- column ruling
- end column name
- row ruling
- span name
- start column name
- vertical alignment

9.Used at the top level

```
fm property prop value is "val";
```

9.1 Arguments

gi A markup element's name (generic identifier).

fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

- This rule applies only to an element corresponding to a cross-reference, graphic, equation, marker, table, or table part element.
- Some FrameMaker properties have no natural markup counterparts. If you choose to not translate such properties as markup attributes, a markup document will not contain information on appropriate values for these properties. In this situation, you can use the `fm property` rule to explicitly set property values when reading a markup document.
- This rule can be used either at the highest level to set a default or within an `element` rule to specify the translation of a property for a particular element.
- If you use this rule to set a property value explicitly, you cannot also have a markup attribute that corresponds to this property. For example, the following rule is erroneous:

```
element "tab2" {
   is fm table element;
   attribute "w" is fm property column widths;
   fm property column widths value is "1in 2in";
}
```

Examples

- To translate the markup element `table` to a FrameMaker table with two columns:

```
element "table" {
   is fm table element;
   fm property columns value is "2";
}
```

  On import to FrameMaker, the software creates the table as a 2-column table in FrameMaker.

- Assume you have a markup element `halfpage` that holds a 4.5 inch by 6.5 inch graphic object; it does not use an attribute to store the size information. You can translate this to a FrameMaker graphic as follows:

```
element "halfpage" {
   is fm graphic element;
   fm property width value is "6.5";
   fm property height value is "4.5";
}
```

See also

Related rules	"is fm property" on page 116 "is fm property value" on page 126
General information on this topic	Developer Guide, Chapter 22, Translating Tables Developer Guide, Chapter 23, Translating Graphics and Equations Developer Guide, Chapter 24, Translating Cross-References Developer Guide, Chapter 26, Translating Markers

fm variable

On export, use the `fm variable` rule to tell FrameMaker what to do with certain variables. Use this rule if you do not want them translated to entities. Read/write rules do not support double-byte characters, so you cannot use this rule to process variables with double-byte characters in their names.

Synopsis and contexts

```
fm variable ["var1", . . ., "varn"] drop;
```

Arguments

var_i	The name of a FrameMaker variable.

Details

- Use this rule when you export FrameMaker documents to markup documents. It applies only to non-element variables, not to system variable elements.
- If var_i is specified, this rule applies only to that variable. If no var_i is specified, this rule applies to all variables.
- If you use this rule, you may want to write a structure API client, or use an XSLT transform to handle the export of variables or to create variables on import.
- You can have multiple occurrences of this rule in a rules document to determine different treatment for different FrameMaker variables. You can have only one occurrence of the rule with no explicitly listed variables. A given variable can be explicitly mentioned in only one occurrence of this rule.

Examples

To translate the FrameMaker variables `Licensor` and `Product` as entities and discard all other variables, use these rules:

```
entity "licensor" is fm variable;
entity "product" is fm variable;
fm variable drop;
```

See also

Related rules	"is fm system variable element" on page 132
General information on this topic	Developer Guide, Chapter 21, Translating Entities and Processing Instructions Developer Guide, Chapter 25, Translating Variables and System Variable Elements *Structure Import/Export API Programmer's Guide*

fm version

The `fm version` rule specifies the version of the product being run. It is required and must be the first rule in all rules documents. If you create your rules document with the New Read/Write Rules command, this rule automatically appears in the document.

Synopsis and contexts

```
fm version is "8.0";
```

ArgumentsNone.

Details

Note that you would use the string `"8.0"` in this rule even though the product version may be an incremental release above 8.0, such as 8.0.1.

See also

General information on this topic	Developer Guide, Chapter 18, Read/Write Rules and Their Syntax

generate book

Use the `generate book` subrule of a highest-level `reader` rule to specify whether FrameMaker should use elements or processing instructions to indicate where in a markup document to start a book and its components in the corresponding FrameMaker book.

Synopsis and contexts

```
1.reader generate book
      use processing instructions;
```

```
2.reader generate book
   {
      put element "gi1" in file ["fname1"];
      . . .
      put element "giM" in file ["fnameM"];
   }
3.reader generate book [for doctype "dt1" [, . . . "dtN"]]
   {
      put element "gi1" in file ["fname1"];
      . . .
      put element "giM" in file ["fnameM"];
   }
```

Arguments

dt_i	A document type name.
gi_j	A generic identifier.
$fname_j$	A filename for the book component. FrameMaker adds a counter to the name (before the suffix if there is one) as needed, to generate a unique filename. You can use the `$(bookname)` variable to base the component's filename on the book filename (excluding any suffix). If you do not supply this argument, the filename is gi_j`.doc`.

Details

- By default, when reading a markup document into FrameMaker, the software uses the `<?FM book ?>` and `<?FM document ?>` processing instructions to indicate the start of a book and of its components. The following rule confirms this default behavior:

  ```
  reader generate book
     use processing instructions;
  ```

- Your DTD may be defined so that you can use elements to indicate the start of a book and its components. When you use the second form of the `generate book` rule, FrameMaker creates a book for every markup document you translate. When you use the third form of the `generate book` rule, it creates a book only for markup documents whose DTD specifies the document type you've listed in the rule. If you have a markup document with a different document type, FrameMaker translates that document as a single FrameMaker document, even if it contains elements referenced in `put element` rules. For example, assume you have this rule:

  ```
  reader generate book for doctype "manual"
     put element "chapter" in file;
  ```

  If you translate a markup document whose highest-level element is `report`, that document becomes a single FrameMaker document, even if it contains `chapter` descendant elements.

- When it encounters one of the gi_j elements specified in a `put element` subrule, FrameMaker starts a new book component. Since the software does not allow an element to be broken across files, it places the entire gi_j element in the same file, even if another element appears that you've said should start a new file. To illustrate, assume the `section` element can occur either within or outside of a `chapter` element and you have this rule:

  ```
  reader generate book {
     put element "chapter" in file;
     put element "section" in file;
  }
  ```

  When FrameMaker encounters a `chapter` element, it starts a new file. If it encounters a `section` element as a child of that `chapter` element, it does not start a new file. It continues with the file started by the `chapter` element. On the other hand, if the software encounters a `section` element outside a `chapter` element it does start a new file for it.

- Consider these points when dividing a markup document into book components:

  –Every FrameMaker document must contain exactly one highest-level element. That is, there cannot be two elements in a single file that do not have an ancestor element in the same file.

  –A book element can contain substructure but cannot directly contain text. That is, child elements that can contain text must occur in separate files.

  Assume you have this rule:

  ```
  reader generate book
     put element "chapter" in file;
  ```

  And you have a markup document with the following element structure:

  ```
  <manual>
  <chapter>
     <head>Introduction</head>
     . . .
  </chapter>
  <appendix>
     <head>The final word</head>
     . . .
  </appendix>
  </manual>
  ```

  When FrameMaker translates this document, it creates a book with `manual` as the highest-level element in the book file. When it encounters the `chapter` element, the software starts a new file for that element. When it encounters the `appendix` element, FrameMaker flags an error, because your rules have not told it what to do with this element. It cannot put the element in the same file as the preceding `chapter` element, because that would create two highest-level elements in the same file. It also cannot put the `appendix` element in the book file, because it contains text.

- By default, when it writes a FrameMaker book to markup, the software writes `<?FM book ?>` and `<?FM document ?>` processing instructions for the book and book components. It does this even if you use the `generate book` rule to have particular elements specify book components when reading a markup document. If you do not want FrameMaker to output these processing instructions, use `writer do not output book processing instructions`.

Examples

- If you know that a markup document should always correspond to a FrameMaker book and that individual files in the book should start when the document reaches a `toc` or `chapter` element, you can use this rule:

```
reader generate book {
   put element "toc" in file;
   put element "chapter" in file "ch.doc";
}
```

  With this rule, FrameMaker creates a book for each markup document. In a markup document, FrameMaker starts a new book component when it encounters a `toc` or `chapter` element. For the first `toc` element, FrameMaker uses the filename `toc1` unless a file of that name already exists in the directory it is using. It continues that book component until it encounters either another `toc` element or a `chapter` element. At that point, it starts a new book component. It tries to put the first `chapter` element in a file called `ch1.doc`.

- Assume that a markup document whose highest-level element is either `manual` or `book` should correspond to a FrameMaker book and any other markup document should correspond to an individual FrameMaker document. Further assume that the books created from `manual` and `book` elements should have new files for each instance of the elements `chapter`, `front`, or `toc`. To accomplish all this, you can use this rule:

```
reader generate book for doctype "manual", "book"
{
   put element "chapter" in file "ch.doc";
   put element "front" in file;
   put element "toc" in file "$(bookname).toc";
}
```

  With this rule, FrameMaker asks you for a name for the book file if you open a markup document with `manual` as its document type. If you specify `myfile.book` as its name, and the document contains two `chapter` elements, one `front` element, and one `toc` element, FrameMaker creates the following files: `myfile.book`, `ch1.doc`, `ch2.doc`, `front`, and `myfile.toc`.

See also

Related rules "output book processing instructions" on page 147

General information on this topic — Developer Guide, Chapter 28, Processing Multiple Files as Books

implied value is

Use the `implied value is` rule to specify default attribute values in your EDD to correspond with imported elements that specify no value for the attribute. For example, assume your DTD declares an element named `list`, which has an attribute named `style` defined as `<!ATTLIST list style (bul | num) #IMPLIED>`. For importing the DTD, you can use this rule to set up a default value in the EDD for the `Style` attribute of the `List` element. Then, if you import a `list` element that has no value for style, this default attribute value will be used for formatting purposes. Also, when you export the EDD, the DTD will declare the `style` attribute for the `list` element as `#IMPLIED`.

Synopsis and contexts

```
1.attribute "attr" {. . .
      implied value is "val";
   . . .}
2.element "gi" {. . .
      attribute "attr" {. . .
         implied value is "val";
   . . .} . . .}
```

Arguments

attr	The name of an impliable attribute in markup.
val	A value to use for the *attr* attribute.
gi	A markup element's name (generic identifier).

Details

- This rule is for importing DTDs and exporting EDDs. In FrameMaker, a default attribute value can only be specified in the EDD, so this rule has no effect when importing a markup instance or exporting a FrameMaker document.
- This rule specifically does not supply an attribute value for an element that has no value in the markup instance. It only sets up a default attribute value in the EDD. This default value can be used for formatting by attributes. When you export the document, FrameMaker will not add a value for the attribute to the element's start tag.
- The rule can be used in a highest-level `attribute` rule to specify the value to use for that attribute in any element. Alternatively, it can be used in an `attribute` rule within an `element` rule to specify the value for that element only.

Examples

Assume you have these declarations for a markup element used for cross-references:

```
<!ELEMENT xref EMPTY>
<!ATTLIST xref
    id IDREF #IMPLIED
    format CDATA #IMPLIED>
```

And you have this rule:

```
element "xref" {
   is fm cross-reference element;
   attribute "format" {
      is fm property cross-reference format;
      implied value is "Page";
}}
```

When FrameMaker encounters an instance of the `xref` element in a markup document and that instance does not have a value for the `format` attribute, the software use the Page cross-reference format for the cross-reference in the FrameMaker document.

See also

Related rules	"value" on page 165
Rules mentioned in synopses	"attribute" on page 46 "element" on page 56
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes "Default value" on page 198

include dtd

By default, when creating a markup document, FrameMaker includes in the document type definition an external identifier that refers to the DTD file. Therefore, it does not include a copy of actual declarations in the document type declaration subset. The `include dtd` rule tells FrameMaker to do so.

Synopsis and contexts

```
writer [do not] include dtd;
```

ArgumentsNone.

Details

- You use this rule when you export FrameMaker documents to markup documents. If this rule is specified, FrameMaker does not generate an external identifier in the `DOCTYPE` declaration.

- To confirm the default behavior, you can use the opposite rule:

```
writer do not include dtd;
```

- The `include dtd` rule and the `external dtd` rule are mutually exclusive. That is, you cannot use both of these rules in the same read/write rules file. (If you try to put both of these rules in the same file, you will get an alert.) Also, the `include dtd` rule and the `write structure document instance only` rule are mutually exclusive.
- To write an entire markup document, including a DTD and (for SGML) an SGML declaration with the document instance, you must use the following rules:

```
writer {
include sgml declaration;
include dtd;
}
```

- This rule uses the DTD that is specified in the current structure application. If that DTD includes references to external files, this rule does not expand those references as it writes out the DTD. Instead, it writes out the references as they appear in the parent DTD file.
- You can use this rule to translate the EDD from the current document as an a DTD that is written in the markup document. To do this, use the `include dtd` rule, but use a structure application that does not specify a DTD in its definition. Be warned that if your document uses the CALS table model, the resulting DTD may be incorrect.
- When you use this rule, no Schema information is included in the output. If you use this rule to output an internal DTD and the XML structure application specifies a Schema file for export, that file is converted to internal DTD (see Chapter 7, "XML Schema to DTD Mapping") and that DTD is saved with the markup document.

  If the XML structure application specifies both a Schema file and a DTD, the DTD is output as the internal DTD and the Schema is dropped.

  If the XML structure application specifies neither a Schema file nor a DTD, an internal DTD is created from the first of these sources that is available:

  –an external DTD for the imported document;

  –a DTD that is the result of conversion from a Schema in the imported document;

  –the element catalog of the template.

Examples

If your document type declarations are in a file called `report.dtd`, then by default FrameMaker includes this document type declaration in the document it creates on export:

```
<!DOCTYPE report SYSTEM "report.dtd" [
. . . more declarations specific to this document instance . . .
]>
```

If you specify the `include dtd` rule, then FrameMaker includes this document type declaration in the document it creates:

```
<!DOCTYPE report [
. . . declarations specific to this document instance . . .
. . . contents of the file, report.dtd . . .
]>
```

See also

Related rules

"external dtd" on page 73
"include sgml declaration," next
"write structured document" on page 166
"write structured document instance only" on page 167

include sgml declaration

By default, FrameMaker does not include an SGML declaration in a generated SGML document. The `sgml declaration` rule tells FrameMaker to include one. The SGML declaration is copied from the file in the associated application subset. To see the default SGML declaration used by FrameMaker, see Chapter 10, "SGML Declaration."

> **Note: XML:** This read/write rule is for SGML-only.

Synopsis and contexts

```
writer [do not] include sgml declaration;
```

ArgumentsNone.

Details

- To confirm the default behavior, you can use the opposite rule:

  ```
  writer do not include sgml declaration;
  ```

- You cannot use the `include sgml declaration` rule in the same read/write rules file as the `write sgml document instance only` rule. Note that using both rules in the same rules file does not give an error. Also, "write sgml document instance only" takes priority, regardless of order.
- To write an entire SGML document, including an SGML DTD and SGML declaration with the document instance, you must use the following rules:

  ```
  writer {
  include sgml declaration;
  include dtd;
  }
  ```

See also

Related rules

"external dtd" on page 73
"include dtd," (the previous section)
"write structured document" on page 166
"write structured document instance only" on page 167

insert table part element

You use the `insert table part element` rule when creating a FrameMaker table element on import of a markup document. This rule tells FrameMaker to create a table part of the designated type, even if the markup document does not contain content for that table part.

Synopsis and contexts

```
element "gi" {. . .
   is fm table element ["fmtag1"];
   reader insert table part element ["fmtag2"];
. . .}
```

Arguments

gi	A markup element's name (generic identifier).
	fmtag$_1$
	A FrameMaker element tag for a table element. These names are case-sensitive and must appear in the rule the same as in the EDD.
part	One of the keywords: `title`, `heading`, or `footing`.
fmtag$_2$	A FrameMaker element tag for a table part element.

Details

By default, as the last step in creating a table element when reading a markup document, FrameMaker discards parts of the table that have no content, even if the general rule for the element requires that table part. (Your EDD may supply the content, for example, by using format rules that specify a prefix for the element.) If you do not want FrameMaker to remove the table part element with no content, OR if you want FrameMaker to create a table part element for you when the markup instance does not contain this element, use the `insert table part element` rule.

Examples

Assume you have a markup element `statetab`, which you represent as a 3-column table in FrameMaker, with the same table headings everywhere it occurs. You use formatting rules in the EDD to specify the table headings. In this situation, the markup document does not include

information that corresponds to the table headings, so you want the software to add the table heading element when reading such a markup instance and drop it when exporting a FrameMaker document to markup. Suppose your DTD has these declarations:

```
<!ELEMENT statetab ((state, pop, income)+)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT pop (#PCDATA)>
<!ELEMENT income (#PCDATA)>
```

and your EDD has these FrameMaker element definitions:

Element (Table): State Table
General rule: State Head, State Body
Text format rules
1. In all contexts.
Use paragraph format: TableCell

Element (Table Heading): State Head
General rule: State Head Row
Text format rules
1. In all contexts.
Default font properties
Weight: Bold

Element (Table Row): State Head Row
General rule: Label

Element (Table Cell): Label
General rule: <EMPTY>
Text format rules
1. If context is: {first}
Numbering properties
Autonumber format: State
Else if context is: {last}
Numbering properties
Autonumber format: Household Income
Else
Numbering properties
Autonumber format: Population

Element (Table Body): State Body
General rule: State Row+

Element (Table Row): State Row
General rule: State, Income, Population

Element (Table Cell): State
General rule: <TEXT>

Element (Table Cell): Income
General rule: <TEXT>

Element (Table Cell): Population
General rule: <TEXT>

Note that the Label element provides the text for the column headings.

You could use these rules:

```
element "statetab" {
   is fm table element "State Table";
   fm property columns value is "3";
   reader insert table heading element "State Head";
}

element "state" {
   is fm table cell element;
   fm property column number value is "1";
   fm property row type value is "Body";
}

element "income" is fm table cell element;

element "pop" is fm table cell element "Population";

fm element "State Head" drop;
fm element "State Body" unwrap;
fm element "State Row" unwrap;
```

To convert the following instance to the desired FrameMaker document:

```
<statetab>
<state>Georgia</state>
<pop>15,000,000</pop>
<income>25,000</income>
<state>Mississippi</state>
<pop>8,000,000</pop>
<income>18,000</income>
</statetab>
```

- The first rule identifies `statetab` as a 3-column table element and tells it to always create a heading element for an occurrence of this `statetab`.
- The second rule identifies `state` as a table cell that must always occur in the first column of a body row. This ensures that FrameMaker starts a new table row whenever it encounters a `state` element.
- The other element rules identify other elements used as table cells. The `fm element drop` rule causes the software to drop the element that was created by FrameMaker per the `insert element` rule so that it does not appear in the markup. Note also that it is necessary for the

software to have a `tablerow` element and a `tablebody` element in its table structure. However, these do not appear in the markup document. FrameMaker creates such necessary elements by default. Since they do not correspond to markup elements, they are unwrapped on export to markup—not dropped, because that would lose the contents of the entire table.

See also

General information on this topic — Developer Guide, Chapter 22, Translating Tables

is fm attribute

Use the `is fm attribute` rule to specify that a markup attribute translates to a FrameMaker attribute. The optional parts of this rule allow you to have the software make several changes to the attribute during translation. Read/write rules do not support double-byte characters, so you cannot use this rule to process attributes with double-byte characters in their names.

Synopsis and contexts

```
1.[mdv] attribute "attr" {. . .
      is fm [read-only] [fmtype] attribute
         ["fmattr"] [range from low to high];
   . . .}

2.element "gi" {. . .
      [mdv] attribute "attr"
         is fm [read-only] [fmtype] attribute
            ["fmattr"] [range from low to high];
   . . .}
```

Arguments

mdv — An optional markup declared value, specifying the type of the markup attribute. Legal values for an XML application are:

- `cdata`
- `nmtoken`
- `nmtokens`
- `entity`
- `entities`
- `id`
- `idref`
- `idrefs`

- `notation`
- `group`

Legal values for an SGML application are:

- `cdata`
- `name`
- `names`
- `nmtoken`
- `nmtokens`
- `number`
- `numbers`
- `nutoken`
- `nutokens`
- `entity`
- `entities`
- `notation`
- `id`
- `idref`
- `idrefs`
- `group`.

attr — A markup attribute name.

fmtype — A FrameMaker attribute type. Legal values are: String, Strings, Integer, Integers, Real, Reals, UniqueID, IDReference, IDReferences, and Choice.

fmattr — A FrameMaker attribute name.

low — A number, indicating the low end of a numeric range.

high — A number, indicating the high end of a numeric range.

Details

- You can use the `is fm attribute` rule in a highest-level `attribute` rule to specify the translation of that attribute in all elements for which it is defined. Or you can use it in an `attribute` subrule in an `element` rule to specify the translation of the `attribute` in only that element.

- You may want some markup attributes to become FrameMaker properties. If so, you cannot also import them as FrameMaker attributes. For information on the defined FrameMaker properties, see "is fm property" on page 116.
- To specify only that the attribute is an attribute in both representations, use this version:

```
attribute "attr" is fm attribute;
```

- To also rename it during translation, use this version:

```
attribute "attr" is fm attribute "fmattr";
```

- To specify that the FrameMaker attribute is read-only—that is, that an end user cannot change the attribute's value—use this version:

```
attribute "attr" is fm read-only attribute;
```

- To specify that an attribute that takes numeric values can have values only in a particular range, use this version:

```
attribute "attr" is fm attribute range from low to high;
```

- To specify that a markup attribute with a particular declared value translates to a FrameMaker attribute of a type other than the default translation, use this version:

```
mdv attribute "attr" is fm fmtype attribute;
```

- To specify that the FrameMaker `direction` property maps to the structured document `dir` attribute, use this version:

```
attribute "dir" is fm attribute; is fm property direction;
```

- Note that you can use more than one of the optional pieces of the `is fm attribute` rule at the same time. For example, you can both rename an attribute and state that it is read-only by using this version:

```
attribute "attr" is fm read-only attribute "fmattr";
```

Examples

- To translate the markup `sec` attribute to the FrameMaker `SecurityRanking` attribute in all elements in which it occurs, use this rule:

```
attribute "sec" is fm attribute "SecurityRanking";
```

- To translate the markup `sec` attribute to the FrameMaker `SecurityRanking` attribute in most elements in which it occurs, but to change it to the `Section` attribute in the `BookPart` element, use these rules:

```
element "BookPart"
   attribute "sec" is fm attribute "Section";

attribute "sec" is fm attribute "SecurityRanking";
```

- Assume you have a markup attribute named `perc` with a declared value of `CDATA`, and assume you know that this attribute always has values that are integers in the range from 0 to 100. You can translate the `perc` attribute to the `Percentage` attribute with this rule:

```
cdata attribute "perc"
   is fm integer attribute "Percentage" range from 0 to 100;
```

- Assume that a markup element has an attribute with declared value `name` and that the attribute has a defined set of allowable values. You can translate that attribute and some of its possible values with the following rule:

```
element "fish" {
   name attribute "loc" {
      is fm choice attribute "CommonLocation";
      value "micro" is fm value "Micronesia";
      value "galap" is fm value "Galapagos Islands";
      value "png" is fm value "Papua New Guinea";
}}
```

See also

Related rules	"fm attribute" on page 76
Rules mentioned in synopses	"attribute" on page 46 "element" on page 56
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes

is fm char

For SGML, use the `is fm char` rule to translate an SGML `SDATA` entity to a single character in FrameMaker. For XML, use this rule to translate an internal entity to a single character in FrameMaker.

Synopsis and contexts

```
1.entity "ename" is fm char ch [in "fmchartag"];
2.reader entity "ename" is fm char ch [in "fmchartag"];
```

Arguments

ename	An entity name.
ch	A one-character string or a numeric character code (specified using the syntax for an octal, hexadecimal, or decimal number described in Developer Guide, page 278: Strings and constants). Note that if the

desired character is a digit or a white-space character, you must enter it as a numeric character code.

fmchartag — A FrameMaker character format tag.

Note that the character format must use a non-standard font family such as Symbol or Zapf Dingbats for this argument to take effect.

Details

- For SGML, instead of using this rule to translate an `SDATA` entity, you can use a parameter literal of a particular form. For information on how to do so, see Developer Guide, page 328: Translating SDATA entities as special characters in FrameMaker.
- For XML, `SDATA` entities are not allowed. This rule translates internal entities to FrameMaker characters, and it translates FrameMaker to internal entities.
- You can use the `is fm char` rule within an `entity` rule at the highest level to have the translation occur in both directions. Or you can put the `entity` rule inside a `reader` rule to have the translation occur only when reading a markup document into FrameMaker. For example, your SGML document might use a `period` entity for entering some instances of the period character in your SGML document. If you use this rule:

  ```
  entity "period" is fm char ".";
  ```

  then the entity references for period in the instance are translated correctly to the period character in FrameMaker. But on export, all periods in the document become references to the `period` entity (which is not likely what you had in mind). To have the period entities read correctly when importing an instance, but have periods remain the period character on export, use this version of the rule:

  ```
  reader
     entity "period" is fm char ".";
  ```

- Without the `in` clause, the software translates the entity using the default character format of the enclosing paragraph element. Frequently, however, special characters require a font change. In these cases, you use the `in` clause.
- For SGML, DTDs frequently use the entity sets defined in Annex D of the SGML Standard, often called ISO public entity sets, for providing commonly used special characters. FrameMaker includes copies of these entity sets and provides rules to handle them for your application. For information on how FrameMaker supports ISO public entities, see Chapter 11, "ISO Public Entities."

Examples

- To translate the `SDATA` entity `sum` as the mathematical summation sign in the Symbol font (S), you could use either of these rules in your rules document:

  ```
  entity "sum" is fm char "S" in "Symbol";
  ```

  ```
  entity "sum" is fm char "\x53" in "Symbol";
  ```

```
entity "sum" is fm char 0x53 in "Symbol";
```

If FrameMaker encounters a reference to the `summation` entity when importing a markup document, it replaces the reference with S (assuming your FrameMaker template defines the Symbol character format appropriately and the entity is declared in the DTD). If the software encounters S when exporting an document, it generates a reference to the `summation` entity (assuming the Symbol character format is defined appropriately and applied to the character, and that the DTD for your application has an entity declaration for “sum”).

- To translate both the `thin` and `en` internal entity references in an XML instance to en spaces in FrameMaker and to write all en spaces as an `en` entity reference, use these rules:

```
entity "en" is fm char 0x13;
reader entity "thin" is fm char 0x13;
```

See also

Rules mentioned in synopses	“entity” on page 61
General information on this topic	Developer Guide, Chapter 21, Translating Entities and Processing Instructions

is fm cross-reference element

Use the `is fm cross-reference element` rule to identify a markup element that translates to a cross-reference element in FrameMaker. You can choose either to have the same name in both representations or to change the name during translation. The markup element should have an attribute of type IDREF and declared content of EMPTY. Read/write rules do not support double-byte characters, so you cannot use this rule to process elements with double-byte characters in their names.

Synopsis and contexts

```
element "gi" {. . .
   is fm cross-reference element ["fmtag"];
. . .}
```

Arguments

gi	A markup element’s name (generic identifier).
	fmtag A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

If you use the `is fm cross-reference element` rule, the other subrules of the `element` rule that you can use for that markup element are as follows:

- `attribute` specifies what to do with a markup element's attributes.
- `fm attribute` specifies what to do with attributes present in the FrameMaker representation of the element but not in the markup representation.
- `fm property` specifies what to do with FrameMaker properties associated with the element.
- `reader drop content` specifies that the content but not the structure of an element should be discarded on import of a markup document.

Examples

- To have the markup element `xref` become the FrameMaker cross-reference element `Xref`, use this rule:

```
element "xref" is fm cross-reference element;
```

- To have it become the FrameMaker cross-reference element `CrossRef`, use this rule:

```
element "xref" is fm cross-reference element "CrossRef";
```

See also

Rules mentioned in synopses	"element" on page 56
General information on this topic	Developer Guide, Chapter 24, Translating Cross-References

is fm element

If you do not specify a value for *fmtag*, the `is fm element` rule specifies only that a markup element remains an element in FrameMaker. This is the default behavior. With a value for *fmtag*, this rule changes the element name when it is translated between markup and FrameMaker.

Synopsis and contexts

```
element "gi" {. . .
   is fm element ["fmtag"];
. . .}
```

Arguments

gi	A markup element's name (generic identifier).
	fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

If you use the `is fm element` rule, the other subrules of the `element` rule that you can use for that markup element are as follows:

- `attribute` specifies what to do with a markup element's attributes.
- `fm attribute` specifies what to do with attributes present in the FrameMaker representation of the element but not in the markup representation.
- `fm property` specifies what to do with FrameMaker properties associated with the element.
- `reader drop content` specifies that the content but not the structure of an element should be discarded on import of a markup document.
- `writer drop content` specifies that the content but not the structure of an element should be discarded on export of a FrameMaker document.

XSLT interaction

XSLT allows precise, context based control over element renaming. For consistency and maintainabilty try to avoid mixing the methods used to rename FrameMaker or XML elements.

Examples

To translate the markup element `par` to the FrameMaker element `Paragraph`, use this rule:

```
element "par" is fm element "Paragraph";
```

See also

Rules mentioned in synopses	"element" on page 56
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes

is fm equation element

Use the `is fm equation element` rule to identify a markup element that translates to an equation element in FrameMaker. You can choose either to have the same name in both representations or to change the name during translation. Read/write rules do not support

double-byte characters, so you cannot use this rule to process elements with double-byte characters in their names.

Synopsis and contexts

```
element "gi" {. . .
   is fm equation element ["fmtag"];
. . .}
```

Arguments

gi A markup element's name (generic identifier).

fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

If you use this rule, the other subrules of the `element` rule that you can use for the same markup element are as follows:

- `attribute` specifies what to do with a markup element's attributes.
- `fm attribute` specifies what to do with attributes present in the FrameMaker representation of the element but not in the markup representation.
- `fm property` specifies what to do with FrameMaker properties associated with the element.
- `writer equation` tells FrameMaker what to do with equation elements.

XSLT interaction

XSLT is not able to convert markup elements to/from FrameMaker equation elements. However, XSLT allows precise, context based control over element renaming. For consistency and maintainabilty try to avoid mixing the methods used to rename FrameMaker or XML elements.

Examples

- To have FrameMaker equation element `Eqn` become the markup element `eqn`, use this rule:

  ```
  element "eqn" is fm equation element;
  ```

- To have FrameMaker equation element `Equation` become the markup element `eqn`, use this rule:

  ```
  element "eqn" is fm equation element "Equation";
  ```

See also

Related rules "is fm graphic element" on page 114

Rules mentioned in synopses	"element" on page 56
General information on this topic	Developer Guide, Chapter 23, Translating Graphics and Equations

is fm footnote element

Use the `is fm footnote element` rule to identify a markup element that translates to a footnote element in FrameMaker. You can choose either to have the same name in both representations or to change the name during translation. Read/write rules do not support double-byte characters, so you cannot use this rule to process elements with double-byte characters in their names.

Synopsis and contexts

```
element "gi" {. . .
   is fm footnote element ["fmtag"];
. . .}
```

Arguments

gi	A markup element's name (generic identifier).
fmtag	A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

If you use this rule, the other subrules of the `element` rule that you can use for the same markup element are as follows:

- `attribute` specifies what to do with a markup element's attributes.
- `fm attribute` specifies what to do with attributes present in the FrameMaker representation of the element but not in the markup representation.

XSLT interaction

XSLT is not able to convert markup elements to/from FrameMaker footnote elements. However, XSLT allows precise, context based control over element renaming. For consistency and maintainabilty try to avoid mixing the methods used to rename FrameMaker or XML elements.

Examples

- To translate the markup element `fn` to the `Fn` footnote element in FrameMaker, use this rule:

```
element "fn" is fm footnote element;
```

- To translate it to the `Footnote` footnote element, use this rule:

```
element "fn" is fm footnote element "Footnote";
```

See also

Rules mentioned in synopses	"element" on page 56
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes

is fm graphic element

Use the `is fm graphic element` rule to identify a markup element that translates to a graphic element in FrameMaker. You can choose either to have the same name in both representations or to change the name during translation. Read/write rules do not support double-byte characters, so you cannot use this rule to process elements with double-byte characters in their names.

Synopsis and contexts

```
element "gi" {. . .
   is fm graphic element ["fmtag"];
. . .}
```

Arguments

gi

A markup element's name (generic identifier).

fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

If you use this rule, the other subrules of the `element` rule that you can use for the same markup element are as follows:

- `attribute` specifies what to do with a markup element's attributes.
- `fm attribute` specifies what to do with attributes present in the FrameMaker representation of the element but not in the markup representation.
- `fm property` specifies what to do with FrameMaker properties associated with the element.
- `writer anchored frame` tells FrameMaker what to do with graphic elements other than those with a single non-internal FrameMaker facet.

- `writer facet` tells FrameMaker what to do with an imported graphic element that has a single non-internal FrameMaker facet.

XSLT interaction

XSLT is not able to convert markup elements to/from FrameMaker graphic elements. However, XSLT allows precise, context based control over element renaming. For consistency and maintainabilty try to avoid mixing the methods used to rename FrameMaker or XML elements.

Examples

- To translate the markup element `pict` to the `Pict` graphic element in FrameMaker, use this rule:

```
element "pict" is fm graphic element;
```

- To translate it to the `Picture` graphic element, use this rule:

```
element "pict" is fm graphic element "Picture";
```

See also

Related rules	"is fm equation element" on page 111
Rules mentioned in synopses	"element" on page 56
General information on this topic	Developer Guide, Chapter 23, Translating Graphics and Equations

is fm marker element

Use the `is fm marker element` rule to identify a markup element that translates to a marker element in FrameMaker. You can choose either to have the same name in both representations or to change the name during translation. Read/write rules do not support double-byte characters, so you cannot use this rule to process elements with double-byte characters in their names.

Synopsis and contexts

```
element "gi" {. . .
   is fm marker element ["fmtag"];
. . .}
```

Arguments

gi	A markup element's name (generic identifier).
	fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

If you use this rule, the other subrules of the `element` rule that you can use for the same markup element are as follows:

- `attribute` specifies what to do with a markup element's attributes.
- `fm attribute` specifies what to do with attributes present in the FrameMaker representation of the element but not in the markup representation.
- `fm property` specifies what to do with FrameMaker properties associated with the element.
- `marker text is` specifies whether the text of a FrameMaker marker element should be element content or an attribute value in markup.

XSLT interaction

XSLT is not able to convert markup elements to/from FrameMaker marker elements. However, XSLT allows precise, context based control over element renaming. For consistency and maintainabilty try to avoid mixing the methods used to rename FrameMaker or XML elements.

Examples

- To translate the markup element `m` to the `M` marker element in FrameMaker, use this rule:

```
element "m" is fm marker element;
```

- To translate it to the `Marker` marker element, use this rule:

```
element "m" is fm marker element "Marker";
```

See also

Related rules	"marker text is" on page 143 "fm marker" on page 79
Rules mentioned in synopses	"element" on page 56
General information on this topic	Developer Guide, Chapter 26, Translating Markers

is fm property

Use the `is fm property` rule to translate a markup attribute to a FrameMaker property. This rule can apply in a highest-level `attribute` rule to set a default. Or it can apply within an `element` rule for a table, table part, marker, cross-reference, graphic, or equation element, to set

the property only for that element. Read/write rules do not support double-byte characters, so you cannot use this rule to process attributes with double-byte characters in their names.

Synopsis and contexts

```
1.attribute "attr" {. . .
      is fm property prop;
   . . .}
2.element "gi" {. . .
      attribute "attr" {. . .
         is fm property prop;
   . . .} . . .}
```

Arguments

attr The name of a markup attribute.

gi A markup element's name (generic identifier).

prop A FrameMaker property. Possible properties are:

- For cross-reference elements:

  `-cross-reference format`

  `-cross-reference id`

- For graphic and equation elements:

  `-alignment` Indicates the anchored frame's horizontal alignment on the page.

  `-angle` Indicates an angle of rotation for the anchored frame that contains the graphic. The markup must specify exact multiples of 90 degrees. Otherwise, the value is ignored and the graphic is imported at 0 degrees which is the default. Examples:

  `-0` No rotation (default).

  `-90` Rotate 90 degrees clockwise.

  `--90` Rotate 90 degrees anticlockwise.

  `-180` Rotate 180 degrees.

  `-270` Rotate 270 degrees.

  `-baseline offset` Indicates how far from the baseline of a paragraph to place an anchored frame. Baseline offset is relevant only for anchored frames whose position attribute is one of inline, sleft, sright, snear, or sfar.

  `-cropped` Indicates whether a wide graphic should be allowed to extend past the margins of the text frame. The `cropped` property is relevant only for anchored frames whose `position` attribute is one of `top`, `below`, or `bottom`.

`-dpi` Indicates how to scale an imported graphic object.

`-entity` Provides the entity name of the imported graphic.

`-file` Provides the file name of the imported graphic.

`-floating` Indicates whether the graphic should be allowed to float from the paragraph to which it is attached. The `floating` property is relevant only for anchored frames whose `position` property is one of `top`, `below`, or `bottom`.

`-height` Indicates the height of the anchored frame. The height of a single imported graphic object is the sum of the height of the object plus twice the value of the vertical offset property.

`-horizontal offset` Indicates how far the graphic object is offset from the right and left edges of the anchored frame.

`-import angle` Indicates an angle of rotation in degrees for the graphic inside its anchored frame.

`-import by reference or copy` Indicates whether an imported graphic object remains in a separate file or is copied into the FrameMaker document on import from markup.

`-import size` indicates the size of the imported graphic object by specifying a width and height.

`-near-side offset` Indicates how far to set a frame from the text frame to which the frame is anchored. It is relevant only for anchored frames whose position attribute is one of `sleft`, `sright`, `snear`, or `sfar`.

`val` A number plus a valid unit of measure, e.g. "12pt", "10mm". If not supplied, the value is 0.

`-position` Indicates where on the page to put the anchored frame. If not supplied, the value is below. Possible anchoring position values are as follows:

`-inline` At insertion point.

`-top` At top of column.

`-below` Below current line.

`-bottom` At bottom of column.

`-sleft` Outside column - left side.

`-sright` Outside column - right side.

`-snear` Outside column - right side.

`-sfar` Outside column - side closer to the page edge.

`-sinside` Outside column - side closer to the binding.

`-soutside` Outside column - side farther from the binding.

`-tleft` Outside text frame - left side.

–tright Outside text frame - right side.

–tnear Outside text frame - side closer to the page edge.

–tfar Outside text frame - side farther from the page edge.

–tinside Outside text frame - side closer to the binding.

–toutside Outside text frame - side closer to the binding.

–runin Run into paragraph.

–sideways Indicates that the imported graphic will be flipped left to right to give a mirror image.

–vertical offset Indicates how far the graphic object is offset from the top and bottom edges of the anchored frame.

–width Indicates the width of the anchored frame. The value for a single imported graphic object is the sum of the width of the object plus twice the value of the horizontal offset property.

- For marker elements:

  –marker text Provides the text content of the marker.

  –marker type Identifies the type of marker.

- For table elements:

  –column ruling Specifies whether all columns should have ruling on their right side. This property does not specify the style or weight of the ruling. The default ruling is defined by the relevant table format in the structured template.

  –column widths The width of successive columns in the table. On import from markup these widths are reapplied regardless of any changes made by the user. If proportional widths are used, the pgwide attribute or page wide property determines the table overall width.

  –columns The number of columns in the table. This is essential for the correct rendering of the table.

  –page wide This is relevant only to tables whose columns use proportional widths on pages with more than a single column. In this case, the attribute indicates whether the entire table should be the width of the column in which it is anchored, or the width of the overall text frame.

  –row ruling Specifies whether all rows should have ruling on their bottom side. This property does not specify the style or weight of the ruling. The default ruling is defined by the relevant table format in the structured template.

  Expected markup attribute value:

  –0 Rows have no ruling.

–1 Rows have ruling.

–table border ruling Specifies whether the table should have ruling around its outside borders. This property does not specify the style or weight of the ruling. The default ruling is defined by the relevant table format in the structured template.

Expected markup attribute value:

–all Rows have no ruling.

–top Rows have ruling.

–table format Specifies the table format for all instances of the FrameMaker table element.

Expected markup attribute value: A name of a table format that is present in the application's structured template.

- For table cell elements:

–column name Associates a name with a cell in a given column.

–column number Indicates the column number that the cell will start in.

–column ruling Specifies whether the cell should have ruling on its right side. This property does not specify the style or weight of the ruling. The default ruling is defined by the relevant table format in the structured template.

Expected markup attribute value:

–0 Cell has no right side ruling.

–1 Cell has right side ruling.

–end column name Specifies the name of a column that ends a straddle.

–horizontal straddle Specifies the number of columns a straddled cell spans.

–more rows Specifies the number of additional rows a straddled cell spans.

Expected markup attribute value: An integer greater than 1 and no greater than the number of rows in the table part. The total number of rows the cell occupies is more rows+1.

–rotate Indicates how much to rotate the contents of a cell.

Expected markup attribute value: The CALS model restricts this property to a boolean value, where 1 indicates a rotation of 90 degrees anti-clockwise. FrameMaker extends the possible values to allow rotations of 0, 90, 180, and 270 degrees.

–row ruling Specifies whether the cell should have ruling on its bottom side. This property does not specify the style or weight of

the ruling. The default ruling is defined by the relevant table format in the structured template.

Expected markup attribute value:

`-0` Cell has no bottom side ruling.

`-1` Cell has bottom side ruling.

`-span name` Applies a predefined CALS spanspec, starting at this cell.

Expected markup attribute value: A valid `spanspec` name.

`-start column name` Specifies the name of a column that begins a horizontal straddle.

Expected markup attribute value: A valid column name as defined in a colspec.

`-vertical straddle` Specifies the number of rows a straddled cell spans.

Expected markup attribute value: An integer greater than 1 and no greater than the number of rows in the section (heading, body or footing) of the table that contains the starting cell.

`-use fill override` Specifies whether a custom fill percentage for the cell shading overrides the fill percentage specified in the table format.

Expected markup attribute value:

`-0` Cell has no fill override.

`-1` Cell has fill override.

`-fill override` Specifies the fill percentage for the cell shading that overrides the fill percentage in the table format.

Expected markup attribute value: A valid fill percentage for the cell shading.

`-use shading override` Specifies whether a custom color for the cell shading overrides the shading color specified in the table format.

Expected markup attribute value:

`-0` Cell has no shading override.

`-1` Cell has shading override.

`-fill override` Specifies the color for cell shading that overrides the shading color in the table format.

Expected markup attribute value: A valid shading color for the cell shading.

`-use bottom ruling override` Specifies whether the cell bottom ruling overrides the bottom ruling specified in the table format.

Expected markup attribute value:

`-0` Cell has no bottom ruling override.

`-1` Cell has bottom ruling override.

`-bottom ruling override` Specifies the style of the cell bottom ruling that overrides the ruling in the table format.

Expected markup attribute value: A valid style for the cell bottom ruling.

`-use top ruling override` Specifies whether the cell top ruling overrides the top ruling specified in the table format.

Expected markup attribute value:

`-0` Cell has no top ruling override.

`-1` Cell has top ruling override.

`-top ruling override` Specifies the style of the cell top ruling that overrides the ruling in the table format.

Expected markup attribute value: A valid style for the cell top ruling.

`-use left ruling override` Specifies whether the cell left ruling overrides the left ruling specified in the table format.

Expected markup attribute value:

`-0` Cell has no left ruling override.

`-1` Cell has left ruling override.

`-left ruling override` Specifies the style of the cell left ruling that overrides the ruling in the table format.

Expected markup attribute value: A valid style for the cell left ruling.

`-use right ruling override` Specifies whether the cell right ruling overrides the right ruling specified in the table format.

Expected markup attribute value:

`-0` Cell has no right ruling override.

`-1` Cell has right ruling override.

`-right ruling override` Specifies the style of the cell right ruling that overrides the ruling in the table format.

Expected markup attribute value: A valid style for the cell right ruling.

`-angle` Specifies the angle of rotation for the cell that overrides the angle in the table format.

Expected markup attribute value: A valid angle of rotation for the cell.

- For table row elements: `maximum height`, `minimum height`, `row type`, or `row ruling`.

  `-maximum height` Specifies the maximum height for each row in the table.

  Expected markup attribute value: A number plus a valid unit of measure, e.g. "24pt", "15mm". If not supplied, the maximum height of the row is not limited.

  `-minimum height` Specifies the minimum height for each row in the table.

  Expected markup attribute value: A number plus a valid unit of measure, e.g. "12pt", "9mm". If not supplied, the minimum height of the row is not limited.

  `-row type` Sets the row type as heading, body or footing.

  `-row ruling` Specifies whether the cell should have ruling on its bottom side. This property does not specify the style or weight of the ruling. The default ruling is defined by the relevant table format in the structured template.

  Expected markup attribute value:

  `-0` Cell has no bottom side ruling.

  `-1` Cell has bottom side ruling.

  `-row placement` Specifies the row placement in the table.

  Expected markup attribute value: A valid position for the row in the table.

  `-keep with prev` Specifies whether the row is always on the same page as the previous row in the table.

  Expected markup attribute value:

  `-0` Row need not remain on the same page as the previous row.

  `-1` Row is always on the same page as the previous row in the table.

  `-keep with next` Specifies whether the row is always on the same page as the next row in the table.

  Expected markup attribute value:

  `-0` Row need not remain on the same page as the next row.

  `-1` Row is always on the same page as the next row in the table.

- For CALS table colspecs:

  `-cell alignment character`

  `-cell alignment offset`

  `-cell alignment type`

```
-column name
-column number
-column ruling
-column width
-row ruling
-vertical alignment
```

- For CALS table spanspecs:

```
-cell alignment character
-cell alignment offset
-cell alignment type
-column ruling
-end column name
-row ruling
-span name
-start column name
-vertical alignment
```

- For elements:

`direction`: Specifies the direction of an element

Expected markup attribute values:

`-ltr`: Position the content left-to-right

`-rtl`: Position the content right-to-left

`-inherit`: Inherit the position property of the parent element

Details

- If you use the `is fm property` rule to translate a markup attribute to a FrameMaker property, the markup attribute does not also appear as a FrameMaker attribute.
- If you use this rule in a highest-level `attribute` rule, it applies only to elements that have that attribute and are of the appropriate type. For example, if you have these declarations:

```
<!ATTLIST (graphic | table) w CDATA #IMPLIED>
```

and these rules:

```
attribute "w" is fm property width;
element "graphic" is fm graphic element;
element "table" is fm table element;
```

the `w` attribute becomes the `width` property of the `graphic` element but remains an attribute for the `table` element, since tables do not have a `width` property. If you intended `w` to be the column width for tables, you should use these rules:

```
element "graphic" {
   is fm graphic element;
   attribute "w" is fm property width;
}

element "table" {
   is fm table element;
   attribute "w" is fm property column width;
}
```

Examples

- The markup attribute `w` may be used for multiple elements to represent the width of a table's columns. To translate it to the FrameMaker property `column width`:

```
attribute "w" is fm property column width;
```

- To translate the attribute `form` to the cross-reference formatting property `cross-reference format` for the element `xref`:

```
element "xref" {
   is fm cross-reference element;
   attribute "form" is fm property cross-reference format;
}
```

- To translate the attribute `dir` to the direction property for an element:

```
attribute "dir" {
   is fm attribute;
 is fm property direction;
}
```

See also

Related rules	"fm property" on page 80 "is fm property value," next
Rules mentioned in synopses	"element" on page 56 "attribute" on page 46
General information on this topic	Developer Guide, page 345: Formatting properties for tables Developer Guide, page 373: Anchored frame properties Developer Guide, page 375: Other graphic properties Developer Guide, Chapter 26, Translating Markers Developer Guide, Chapter 24, Translating Cross-References

is fm property value

Use the `is fm property value` rule when a markup attribute has a name token group as its declared value and you want to rename the individual name tokens when translating to and from FrameMaker property values. Read/write rules do not support double-byte characters, so you cannot use this rule to process attributes with double-byte characters in their names.

Synopsis and contexts

```
1.value "token" is fm property value propval;

2.attribute "attr" {. . .
      value "token" is fm property value propval;
   . . .}

3.element "gi" {. . .
      attribute "attr" {. . .
         value "token" is fm property value propval;
   . . .} . . .}
```

Arguments

token	A token in a name token group.
propval	A defined FrameMaker property value.
attr	The name of a markup attribute.
gi	A markup element's name (generic identifier).

Details

- This rule can be used at the highest level to set a default, or within an `attribute` rule.
- Use this rule when the corresponding markup attribute translates to a property in FrameMaker. If the markup attribute translates to a choice attribute instead, you need to use the `is fm value` rule to specify the correspondence between markup tokens and FrameMaker attribute choices.

- When using this rule, remember that markup does not permit a token to appear in the declared value of more than one attribute of an element. For example, the following rule:

```
element "picture" {
   is fm graphic element;
   attribute "place" {
      is fm property position;
      value "left" is fm property value subcol left;
    }
   attribute "just" {
      is fm property alignment;
      value "left" is fm property value align left;
}}
```

  corresponds to an erroneous markup ATTLIST such as:

```
<!ATTLIST picture
   place (left, sright, snear, . . .)
   just (left, aright, acenter, . . .)
>
```

- FrameMaker defines the `table border ruling` property for working with tables and the `alignment` and `vertical alignment` properties for working with colspecs and spanspecs.

  If you use the CALS table model for your tables, you should use read/write rules to translate these properties to the `frame`, `align`, and `valign` attributes on appropriate elements. There is also a default correspondence between the FrameMaker property values and the defined value in markup.

  If you do not use the CALS table model, you may still choose to translate these FrameMaker formatting properties to markup attributes. In this case, you must also determine the translation from property value to defined value.

- If you use the CALS table model, the `frame` attribute has the following defined values: `all`, `top`, `bottom`, `topbot`, `sides`, and `none`. The values for the corresponding `table border ruling` property are the same as the defined values, except that the `topbot` defined value is the `top and bottom` property value.

  The `align` attribute and the corresponding `cell alignment type` property have the following values: `left`, `center`, `right`, `justify`, and `char`.

  The `valign` attribute and the corresponding `vertical alignment` property have the following values: `top`, `middle`, and `bottom`.

Examples

- To use the `table border ruling` property for a non-CALS table and to set its name tokens, use this rule:

```
element "tab" {
   is fm table element;
   attribute "frame" {
      is fm property table border ruling;
      value "all" is fm property value all;
      value "top" is fm property value top;
      value "bottom" is fm property value bottom;
      value "topbot" is fm property value top and bottom;
      value "sides" is fm property value sides;
      value "none" is fm property value none;
}}
```

- To rename the FrameMaker `import by reference or copy` property as the `refcopy` attribute, and to also change the name tokens, use this rule:

```
attribute "refcopy" {
   is fm property import by reference or copy;
   value "r" is fm property value reference;
   value "c" is fm property value copy;
}
```

See also

Related rules	"fm property" on page 80
	"is fm property" on page 116
Rules mentioned in synopses	"attribute" on page 46
	"element" on page 56
	"value" on page 165

is fm reference element

For SGML, use the `is fm reference element` rule to translate an entity in markup to an element defined on a reference page in a FrameMaker document (a reference element). For XML, use this rule to translate an internal entity to a reference element. Read/write rules do not support double-byte characters, so you cannot use this rule to process elements with double-byte characters in their names.

Synopsis and contexts

```
1.entity "ename" is fm reference element ["fmtag"];
```

```
2.reader entity "ename" is fm reference element ["fmtag"];
```

Arguments

ename	An entity name.
	fmtag
	A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

- For SGML, instead of using this rule to translate an `SDATA` entity, you can use a parameter literal of a particular form. For information on how to do so, see Developer Guide, page 332: Translating SDATA entities as FrameMaker reference elements.
- For XML, `SDATA` entities are not allowed—this rule translates internal entities.
- You can use the `is fm reference element` rule within an `entity` rule at the highest level to have the translation occur in both directions. Or you can put the `entity` rule inside a `reader` rule to have the translation occur only when reading an SGML document into FrameMaker. Remember that the SDATA entity must be declared in the DTD in order to use this rule.
- The FrameMaker element must occur in a flow named `Reference Elements`. That flow must be on a reference page of the application's template file with a name that starts with `SGML Utilities Page`—for example, `SGML Utilities Page 1` or `SGML Utilities Page Logos`. For information on working with reference pages, see the FrameMaker user guide.
- When FrameMaker encounters references to the specified entity while translating an markup document to FrameMaker, it copies the appropriate element from its reference page in the FrameMaker template associated with the structure application. When it encounters an instance of an element associated with one of the reference pages while writing a FrameMaker document to markup, it generates an entity reference.
- When you use this rule, the *fmtag* element must be defined for your FrameMaker documents and valid in the contexts in which *ename* occurs. If it is not, the resulting FrameMaker document is invalid.

Examples

Assume you have an entity named `legalese` which contains text you need to include in many places. The entity is too long to be a FrameMaker variable, and you don't want to treat it as an entire paragraph. Instead, you can choose to have the entity correspond to a text range element called `LegaleseFragment`.

To do so, add the following rule to your rules document:

```
entity "legalese" is fm reference element "LegaleseFragment";
```

The entity declaration in your DTD looks like this for XML:

```
<!ENTITY legalese "">
```

The entity declaration in your DTD looks like this for SGML:

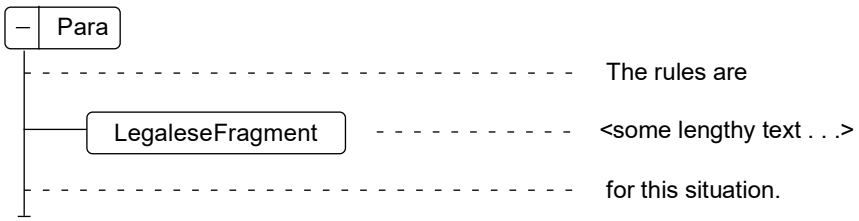
```
<!ENTITY legalese SDATA "[]">
```

Create a reference frame on the reference page of your application which contains the element "LegaleseFragment" with your boilerplate text. In order for the element to be treated as a "text range" use the appropriate TextFormatRules for this element in the EDD.

When FrameMaker translates a markup document that contains the following markup:

```
<para>The rules are &legalese; for this situation.</para>
```

It produces the following element structure:



See also

Rules mentioned in synopses	"entity" on page 61
General information on this topic	Developer Guide, Chapter 21, Translating Entities and Processing Instructions

is fm rubi element

Use the `is fm rubi element` rule to identify a markup element that translates to a Rubi element in FrameMaker. You can choose either to have the same name in both representations or to change the name during translation. Read/write rules do not support double-byte characters, so you cannot use this rule to process elements with double-byte characters in their names.

Synopsis and contexts

```
element "gi" {. . .
   is fm rubi element ["fmtag"];
. . .}
```

Arguments

gi	A markup element's name (generic identifier).

fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

If you use this rule, the other subrules of the `element` rule that you can use for the same markup element are as follows:

- `attribute` specifies what to do with a markup element's attributes.
- `fm attribute` specifies what to do with attributes present in the FrameMaker representation of the element but not in the markup representation.

Examples

- To translate the markup element `rubitext` to the `Rubitext` element in FrameMaker, use this rule:

```
element "rubitext" is fm rubi element;
```

- To translate it to the `MyRubiTextp` element, use this rule:

```
element "rubitext" is fm rubi element "MyRubiText";
```

See also

Rules mentioned in synopses	"element" on page 56
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes

is fm rubi group element

Use the `is fm rubi group element` rule to identify a markup element that translates to a Rubi group element in FrameMaker. You can choose either to have the same name in both representations or to change the name during translation. Read/write rules do not support double-byte characters, so you cannot use this rule to process elements with double-byte characters in their names.

Synopsis and contexts

```
element "gi" {. . .
   is fm rubi group element ["fmtag"];
. . .}
```

Arguments

gi	A markup element's name (generic identifier).

fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

If you use this rule, the other subrules of the `element` rule that you can use for the same markup element are as follows:

- `attribute` specifies what to do with a markup element's attributes.
- `fm attribute` specifies what to do with attributes present in the FrameMaker representation of the element but not in the markup representation.

Examples

- To translate the markup element `rubigroup` to the `Rubigroup` element in FrameMaker, use this rule:

```
element "rubigroup" is fm rubi group element;
```

- To translate it to the `MyRubiGroup` element, use this rule:

```
element "rubigroup" is fm rubi group element "MyRubiGroup";
```

See also

Rules mentioned in synopses	"element" on page 56
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes

is fm system variable element

Use the `is fm system variable element` rule to identify a markup element that translates to a system variable element in FrameMaker. You can choose either to have the same name in both representations or to change the name during translation. Read/write rules do not support double-byte characters, so you cannot use this rule to process elements with double-byte characters in their names.

Synopsis and contexts

```
element "gi" {. . .
   is fm system variable element ["fmtag"];
. . .}
```

Arguments

gi	A markup element's name (generic identifier).

fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

- If you use this rule, the other subrules of the `element` rule that you can use for the same markup element are:

  `attribute` specifies what to do with a markup element's attributes.

  `fm attribute` specifies what to do with attributes present in the FrameMaker representation of the element but not in the markup representation.

- This rule does not apply to translating non-element FrameMaker variables.

Examples

To translate the markup element `date` to the `Date` system variable element in FrameMaker, use this rule:

```
element "date" is fm system variable element;
```

You specify which system variable to use by adding a rule to the `Date` element's definition in the FrameMaker EDD. For example:

Element (System Variable): Date

System variable format rule

In all contexts.
Use system variable: Current Date (Long)

See also

Related rules	"is fm variable" on page 140 "fm variable" on page 92
Rules mentioned in synopses	"element" on page 56
General information on this topic	Developer Guide, Chapter 25, Translating Variables and System Variable Elements

is fm table element

Use the `is fm table element` rule to identify a markup element that translates to a table element in FrameMaker. You can choose either to have the same name in both representations or

to change the name during translation. Read/write rules do not support double-byte characters, so you cannot use this rule to process elements with double-byte characters in their names.

Synopsis and contexts

```
element "gi" {. . .
   is fm table element ["fmtag"];
. . .}
```

Arguments

gi A markup element's name (generic identifier).

fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

- If you use the CALS table model, you do not need to use this rule to translate the CALS `table` element properly.
- If your markup element declarations for a table element do not include an attribute that corresponds to the `columns` property, you must use the `fm property` rule to specify a number of columns for the table.
- If you use this rule, the other subrules of the `element` rule that you can use for the same markup element are as follows:
  - `attribute` specifies what to do with a markup element's attributes.
  - `fm attribute` specifies what to do with attributes present in the FrameMaker representation of the element but not in the markup representation.
  - `fm property` specifies what to do with FrameMaker properties associated with the element.
  - `reader insert table part element` indicates that the software should insert the indicated table part (table title, table heading, or table footing), even if the markup element structure or instance does not contain the corresponding element.

Examples

- To translate the markup element `gloss` to the `Gloss` table element in FrameMaker, use this rule:

```
element "gloss" is fm table element;
```

- To translate it to the `Glossary` table element, use this rule:

```
element "gloss" is fm table element "Glossary";
```

See also

Rules mentioned in synopses	"element" on page 56
General information on this topic	Developer Guide, Chapter 22, Translating Tables

is fm table part element

Use the `is fm table part element` rule to identify a markup element that translates to a table part element in FrameMaker, such as a table title element. You can choose either to have the same name in both representations or to change the name during translation. Read/write rules do not support double-byte characters, so you cannot use this rule to process elements with double-byte characters in their names.

Synopsis and contexts

```
element "gi" {. . .
   is fm table part element ["fmtag"];
. . .}
```

Arguments

gi	A markup element's name (generic identifier).
part	A FrameMaker table part. One of the keywords: `title`, `body`, `heading`, `footing`, `row`, `cell`.
	fmtag
	A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

- If you use the CALS table model, you do not need to use this rule to translate elements representing parts of tables in CALS properly.
- If you map a markup element to a FrameMaker table part element, then the element cannot be used anywhere in the instance except as that table part. For example, if you have a `"title"` element and you use the following rule:

  ```
  element "title" is fm table title element;"
  ```

  Then you would not be able to insert a `"title"` element in a `Chapter` element.
- If you use this rule, the other subrules of the `element` rule that you can use for the same markup element are as follows:

  `–attribute` specifies what to do with a markup element's attributes.

- –fm attribute specifies what to do with attributes present in the FrameMaker representation of the element but not in the markup representation.
- –fm property specifies what to do with FrameMaker properties associated with the element.
- –reader end vertical straddle indicates that the associated table row or cell element terminates a vertical table straddle. This subrule applies only if *part* is row or cell.
- –reader start new row indicates that the associated table cell element indicates the start of a new row in the table. This subrule applies only if *part* is cell.
- –reader start vertical straddle indicates that the associated table cell element starts a vertical table straddle. This subrule applies only if *part* is cell.

Examples

- To translate the markup element head as the FrameMaker table heading element Head, use this rule:

```
element "head" is fm table heading element;
```

- To translate the markup element dfn as the FrameMaker table cell element Definition, use this rule:

```
element "dfn" is fm table cell element;
```

See also

Rules mentioned in synopses	"element" on page 56
General information on this topic	Developer Guide, Chapter 22, Translating Tables

is fm text inset

Use the is fm text inset rule to translate a declared entity to a text inset in FrameMaker. While you can translate any entity to a text inset, we suggest you only do this with SDATA entities when working with SGML. Read/write rules do not support double-byte characters, so you cannot use this rule to process elements with double-byte characters in their names.

> **Note: XML:** The XML standard does not allow SDATA entities, so you cannot use this rule for that purpose. FrameMaker translates external text entities as text insets by default, so this rule is not necessary for that type of entity.

Synopsis and contexts

```
1.entity "ename" is fm text inset "fname"
   [in body_or_ref flow "flowname"];
```

```
2.reader entity "ename" is fm text inset "fname"
   [in body_or_ref flow "flowname"];
```

Arguments

ename	An entity name.
fname	A filename containing the text to include. This file must be a FrameMaker document or a file of a type for which FrameMaker has a filter, for example, a MS-Word document.
body_or_ref	One of the keywords: `body` or `reference`, indicating the type of text flow in which to find the text to include. You can specify this option only if *fname* is a FrameMaker document.
flowname	The name of the FrameMaker text flow.

Details

- By default, external text entities in markup are imported as text insets. For the markup to be valid, the external text entities must be text, XML, or SGML files. In the FrameMaker document, the text insets use these files as their sources. It is probably most advantageous to retain these files for the text insets; you do not need to use the `is fm text inset` rule to import external text entities as text insets.
- The source file for the text inset must either be a FrameMaker file or a file of a format FrameMaker can filter automatically. You cannot use an SGML file as the source of the text inset.
- Instead of using this rule to translate an SGML `SDATA` entity to a text inset, you can use a parameter literal of a particular form. For information on how to do so, see Developer Guide, page 330: Translating SDATA entities as FrameMaker text insets.
- You can use the `is fm text inset` rule within an `entity` rule at the highest level to have the translation occur in both directions. Or you can put the `entity` rule inside a `reader` rule to have the translation occur only when reading an SGML document into FrameMaker.
- If *fname* is not a FrameMaker document, you cannot specify the `in body flow` or `in reference flow` options. In this case, FrameMaker uses all of the text in the file specified by *fname* for the text inset.

  If *fname* is a FrameMaker document and you do not specify a flow to use, FrameMaker use the contents of the main body flow of the specified document.
- **Important:** *flowname* must exactly match the name of a flow in the document. If there is no match for the type of flow you specify (body or reference), then a crash will result. If there is more than one matching flow, FrameMaker uses the first matching flow.
- By default, the software reformats the text inset to conform to the format rules of the document containing the text inset. If the source for the text inset has element structure, FrameMaker also retains that element structure.

You can confirm this behavior with the `reformat using target document catalogs` rule. You can change this behavior using the subrules `reformat as plain text` or `retain source document formatting`.

- FrameMaker requires that a structured flow have exactly one highest-level element. For this reason, you cannot use a single text inset to include multiple elements at the top level of the inset. You must use multiple text insets for this purpose.
- FrameMaker puts an end-of-paragraph symbol after a text inset. For this reason, you cannot use a text inset to insert a range of text inside a single paragraph. To do so, you can translate the entity either as a FrameMaker variable (with the `is fm variable` rule) or as a reference element (with the `is fm reference element` rule).

Examples

Assume you have declared an SGML SDATA entity. You also have a single paragraph of boilerplate text to be used in your documents. You can place this text on a reference page in a text column with a flow called `BoilerPlate` in the FrameMaker template for your SGML application. If that template is the file `template.doc`, you could use this rule to translate occurrences of the `boiler` entity to a text inset in corresponding FrameMaker documents:

```
entity "boiler"
   is fm text inset "template.doc"
   in reference flow "BoilerPlate";
```

See also

Related rules	"reformat as plain text" on page 154 "reformat using target document catalogs" on page 155 "retain source document formatting" on page 156 "is fm reference element" on page 128 "is fm variable" on page 140
Rules mentioned in synopses	"entity" on page 61 "reader" on page 153
General information on this topic	Developer Guide, Chapter 21, Translating Entities and Processing Instructions

is fm value

Use the `is fm value` rule to translate the value of a markup attribute to a particular choice for a FrameMaker choice attribute. The attribute's declared value must be a name token group or `NOTATION`.

Synopsis and contexts

```
1.value "token" is fm value "val";

2.attribute "attr" {. . .
      value "token" is fm value "val";
   . . .}

3.element "gi" {. . .
      attribute "attr" {. . .
         value "token" is fm value "val";
   . . .} . . .}
```

Arguments

token	A token in a name token group.
val	An allowed value for a FrameMaker choice attribute.
attr	The name of a markup attribute.
gi	A markup element's name (generic identifier).

Details

Use this rule when the corresponding markup attribute translates to a choice attribute in FrameMaker. If the markup attribute translates to a FrameMaker property, you need to use the `is fm property value` rule to specify the correspondence between markup tokens and FrameMaker property values.

Examples

- If the token list `(r | b | g)` is used by multiple attributes, you can use these rules to translate the individual tokens consistently:

```
value "r" is fm value "Red";
value "b" is fm value "Blue";
value "g" is fm value "Green";
```

- If the token list `(r | b | g)` is used by several attributes as above but by the `bird` element differently, you can add this rule to the above rules:

```
element "bird" {is fm element;
] attribute "species" {
   value "r" is fm value "Robin";
   value "b" is fm value "Blue Jay";
   value "g" is fm value "Goldfinch";
}}]
```

See also

Related rules	"is fm property value" on page 126
Rules mentioned in synopses	"attribute" on page 46 "element" on page 56 "value" on page 165
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes

is fm variable

Use the `is fm variable` rule to translate a declared markup text entity to a FrameMaker non-element variable.

Synopsis and contexts

`1.entity "`*`ename`*`"` **`is fm variable`** `["`*`var`*`"];`

`2.reader entity "`*`ename`*`"` **`is fm variable`** `["`*`var`*`"];`

Arguments

`ename`	An entity name.
`var`	A FrameMaker variable name.

Details

You can use the `is fm variable` rule within an `entity` rule at the highest level to have the translation occur in both directions. Or you can put the `entity` rule inside a `reader` rule to have the translation occur only when reading a markup document into FrameMaker.

Examples

- To translate the markup element `v` to a non-element FrameMaker variable of the same name:

```
entity "v" is fm variable;
```

- To translate the FrameMaker variable `Licensor` to the markup element `lic`, use this rule:

```
entity "lic" is fm variable "Licensor";
```

See also

Related rules	"fm variable" on page 92 "is fm system variable element" on page 132
Rules mentioned in synopses	"entity" on page 61
General information on this topic	Developer Guide, Chapter 25, Translating Variables and System Variable Elements

is processing instruction

On export, you use the `is processing instruction` rule to tell FrameMaker to create processing instructions for all non-element markers or for non-element markers of a particular type. By default, FrameMaker creates processing instructions for all non-element markers. You have the option of discarding non-element markers; you might use this rule in conjunction with the `drop` rule when you want to discard some but not all non-element markers.

Synopsis and contexts

```
fm marker ["type1", . . ., "typen"] is processing instruction;
```

Arguments

typei	A FrameMaker marker type, such as Index or Type 22.

Details

If you do not supply any *typei* arguments, this rule applies to all non-element markers other than markers of the type reserved by FrameMaker for storing processing instructions, PI entities, and external data entities. (By default, the reserved marker types are `DOC PI`, `DOC Entity Reference`, and `DOC Comment`.)

Examples

To discard all nonelement markers other than Index markers, use these rules:

```
fm marker "Index" is processing instruction;
fm marker drop;
```

See also

Rules mentioned in synopses	"fm marker" on page 79

General information on this topic — Developer Guide, Chapter 26, Translating Markers

line break

Use the `line break` rule to tell FrameMaker about any limits on the length of lines in a markup file it generates. You also use it to tell the software whether or not to interpret line breaks in a markup document as FrameMaker paragraph breaks within elements.

Synopsis and contexts

```
1.reader line break is mode;

2.writer line break is mode;

3.element "gi" {. . .
      reader {. . .
         line break is mode;
   . . .} . . .}

4.element "gi" {. . .
      writer {. . .
         line break is mode;
   . . .} . . .}
```

Arguments

mode	For `writer`: *n* `characters` (where *n* is a positive integer in C syntax). For `reader`: one of `forced return` or `space`.
gi	A markup element's name (generic identifier).

Details

- This rule can be used at the highest level to set a default or within an `element` rule to set line breaks for only that element.
- On export, FrameMaker behaves as follows:

  When exporting the text of a paragraph, it ignores line breaks. It includes a space separating the two words on either side of a line break and attempts to avoid generating lines longer than *n* characters (the default is 80). It maintains a counter indicating how many characters it has placed on a single line. After this counter reaches *n*`-10`, it changes the next data character space to a record end.

  It generates a markup record end at the end of every paragraph and flow in the FrameMaker document.

  If you want a start-tag for an element and its contents to appear on the same line in the markup document, you must write aa structure API client.

- On import you have control over record ends not ignored by the underlying parser. Within a `reader` rule, *`mode`* can be one of the following:

  `forced return` informs FrameMaker that a line break within a text segment should be converted to a forced return.

  `space` informs FrameMaker that a line break within a text segment should be treated as a space. This is the default.

Examples

Line breaks may need to be treated differently within different elements. For example, a line break within an `example` element may need to be preserved on import, while a line break within a `par` element may be a word break:

```
element "example" reader line break is forced return;
element "par" reader line break is space;
```

marker text is

Use the `marker text is` rule to indicate whether the text of a marker element should become an attribute value or the content of the corresponding markup element. Note that the markup element must not be declared as empty if you want the marker text to be translated as content.

Synopsis and contexts

```
element "gi" {. . .
   is fm marker element ["fmtag"];
   marker text is attr_or_content;
. . .}
```

Arguments

`gi`	A markup element's name (generic identifier).
	`fmtag` A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.
`attr_or_content`	One of the keywords: `attribute` or `content`.

Details

- By default, FrameMaker translates a marker element in FrameMaker to a markup empty element. It writes the marker text as the value of the markup element's `text` attribute.
- Instead of the default, you can have FrameMaker translate a marker element to a markup element whose content model is `#PCDATA`. The marker text becomes the element's content.

Examples

- To state that the markup element `mkr` corresponds to the FrameMaker element `Marker` and to confirm the default behavior, you can use this rule:

```
element "mkr" {
   is fm marker element "Marker";
   marker text is attribute;
}
```

With this rule, the FrameMaker element definition:

Element (Marker): Marker

corresponds to the DTD declarations:

```
<!ELEMENT mkr EMPTY>
<!ATTLIST mkr
text CDATA #IMPLIED
type CDATA #IMPLIED>
```

In this case, if the FrameMaker document contains an instance of the `Marker` element whose marker text is “Some marker text” and whose type is Type 22, the markup output includes:

```
<mkr text="Some marker text" type="Type 22"/>
```

- To state that the markup element `mkr` corresponds to the FrameMaker element `Marker` but that the marker text should become element content in markup, you can use this rule:

```
element "mkr" {
   is fm marker element "Marker";
   marker text is content;
}
```

With this rule, the FrameMaker element definition:

Element (Marker): Marker

corresponds to the DTD declarations:

```
<!ELEMENT mkr (#PCDATA)>
<!ATTLIST mkr type CDATA #IMPLIED>
```

In this case, if the FrameMaker document contains an instance of the `Marker` element whose marker text is “Some marker text” and whose type is Type 22, the output includes:

```
<mkr type="Type 22">
Some marker text
</mkr>
```

See also

Rules mentioned in synopses

“element” on page 56
“is fm marker element” on page 115

General information on this topic — Developer Guide, Chapter 26, Translating Markers

notation is

Use the `notation is` rule only in an `element` rule for a graphic or equation element, to provide information the software needs when writing a document containing graphics and equations to markup. FrameMaker uses this rule to determine the data content notation name to include in entity declarations it generates.

Synopsis and contexts

```
1.element "gi" {
      is fm equation element ["fmtag"];
      writer equation notation is "notation";
   . . .}}
2.element "gi" {
      is fm graphic element ["fmtag"];
      writer anchored frame notation is "notation";
   . . .}}
3.element "gi" {
      is fm graphic element ["fmtag"];
      writer facet "facetname" notation is "notation";
   . . .}}
```

Arguments

gi — A markup element's name (generic identifier).

fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

type — One of the rules `anchored frame`, `facet`, or `equation`. If `facet`, you must also supply the *facetname* argument.

If *type* is `equation`, the rule applies to equation elements.

If *type* is `facet`, the rule applies to a graphic element that contains only a single facet with the name specified by *facetname*. This occurs when the graphic element is an anchored frame containing only a single imported graphic object whose original file was in the *facetname* graphic format. You can use this rule with *type* set to `facet` multiple times if you want the software to treat several file formats differently.

If *type* is `anchored frame`, the rule applies to a graphic element under all other circumstances.

facetname

A facet name. The string for the facetname must exactly match the string for the facetname in the FrameMaker document. To determine a graphic file's facetname, select the graphic, click Graphics>ObjectProperties, and observe the facetname in the dialog box.

notation A string representing a data content notation name.

Details

By default, FrameMaker uses the first eight characters of the name of the facet it exports as the data content notation. If the graphic or equation has only internal FrameMaker facets, the software uses CGM as the data content notation.

Examples

Assume your end users use the `af` graphic element within FrameMaker, creating the graphics using FrameMaker tools, but want to store them in TIFF format on export. Furthermore, you want to name the files based on the FrameMaker document's name, but with an extension of `.gr`. You can accomplish this with the following rule:

```
element "af" {
   is fm graphic element;
   writer anchored frame {
      notation is "TIFF";
      export to file "$(docname).gr";
}}
```

If you export the FrameMaker file `intro.doc`, the software writes the following entity declaration for the first instance of the `af` element that it finds:

```
<!ENTITY af1 SYSTEM "intro1.gr" NDATA TIFF>
```

See also

Related rules

"convert referenced graphics" on page 51
"entity name is" on page 63
"export to file" on page 69
"specify size in" on page 157

Rules mentioned in synopses	"element" on page 56 "is fm graphic element" on page 114 "is fm equation element" on page 111 "anchored frame" on page 43 "equation" on page 65 "facet" on page 74 "writer" on page 168
General information on this topic	Developer Guide, Chapter 23, Translating Graphics and Equations

output book processing instructions

By default, when FrameMaker converts a FrameMaker book to markup, it puts `?FM book?` and `?FM document?` processing instructions in the markup document to indicate where the individual files in the FrameMaker documents began. You use the `output book processing instructions` rule to confirm or change this behavior.

Synopsis and contexts

```
writer [do not] output book processing instructions;
```

ArgumentsNone.

Details

If you use the `generate book` rule to tell FrameMaker to use elements to identify book components when reading a markup document, you might choose to not have it output processing instructions when writing the book to markup. In this case, use this rule:

```
writer do not output book processing instructions;
```

See also

Related rules	"generate book" on page 93

preserve fm element definition

Use the `preserve fm element definition` rule to tell FrameMaker, when it is updating an EDD from a revised DTD, not to update the definition of a set of FrameMaker elements and their attributes on the basis of the DTD and other rules.

Synopsis and contexts

```
reader {. . .
   preserve fm element definition "fmtag1"[, . . ., "fmtagN"];
. . .}
```

Arguments

fmtag$_i$

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

FrameMaker uses the `preserve fm element definition` rule only when updating an EDD from a DTD. By default, when it updates an existing EDD, the software changes the definitions of FrameMaker elements to reflect the new DTD and all read/write rules. You may not want the definition of the FrameMaker element to change. For example, if one of your rules is to unwrap the element `body`, then any element with a definition that includes `body` will be modified directly include the contents of `body` instead of including `body`.

Examples

- Assume you have the rule:

  ```
  fm element "Body" unwrap;
  ```

  and the element definitions:

  Element (Container): Figure1
  General rule: Caption, Body

  Element (Container): Figure2
  General rule: Body, Footer

  Element (Container): Body
  General rule: Header, Line+

  The corresponding declarations are:

  ```
  <!ELEMENT figure1 (caption, header, line+)>
  <!ELEMENT figure2 (header, line+, footer)>
  ```

If you update the EDD containing the preceding definitions and use as input the DTD with the preceding declarations, FrameMaker replaces the definitions of `Figure1` and `Figure2` with:

Element (Container): Figure1
General rule: Caption, Header, Line+

Element (Container): Figure2
General rule: Header, Line+, Footer

If you wish to retain the original definitions of `Figure1` and `Figure2` in the revised EDD, include this rule:

```
reader preserve fm element definition "Figure1", "Figure2";
```

- Suppose you want to use a structure API client to reverse the order of child elements in corresponding markup and FrameMaker elements. For example, assume you have the declaration:

  ```
  <!ELEMENT ex (a, b)>
  ```

  and the FrameMaker element definition:

  Element (Container): Ex
  General rule: B, A

  If you have no rules and update the EDD in this situation, FrameMaker updates the definition of `Ex` to correspond to the markup declaration. To suppress this change, use this rule:

  ```
  reader preserve fm element definition "Ex";
  ```

See also

Related rules "drop" on page 53
"unwrap" on page 162

preserve line breaks

Use the `preserve line breaks` rule to tell FrameMaker to keep line breaks for an element when importing and exporting markup documents. When importing markup, it translates every RE in the element as a forced return. When exporting markup, it translates forced returns as RE characters, and the line ends FrameMaker creates when automatically wrapping the text as non-RE line breaks in the markup file. This is useful for elements that use RE characters to insert white space in an element's content.

Synopsis and contexts

```
element {. . .
   preserve line breaks ;
. . .}
```

ArgumentsNone

Details

- For an element using this rule, the software writes a an RE (line break) immediately after the open tag and immediately before the close tag.
- For an element using this rule, on export, FrameMaker writes a space character entity reference and an RE (line break) for each necessary line break in the markup file. See the "line break" rule for information on how FrameMaker determines where to put these line breaks by default. Forced returns (shift-return) translate as RE characters (line breaks) in the markup file.
- For SGML, the space character entity uses the ISO entities reference (`&#SPACE`).
- For XML, no entity reference is written for the space character.
- For XML, this rule adds the `xml:space` attribute to the affected elements, with a value of `preserved`. This attribute directs XML applications to respect the whitespace characters in the element's content. On import this attribute is retained—if the EDD for your template does not specify an `xml:space` attribute for the given element, then that attribute will be invalid. You can either define this attribute in your EDD, or use read/write rules to drop the attribute on import.
- For export and import to have the same results, `preserve line breaks` must be specified for the same elements. For example, assume you use `preserve line breaks` on export for an element named `Code`. FrameMaker writes a space character entity reference and an RE (line break) when a line approaches the maximum line length, and it writes RE characters (line breaks) for forced returns. Now assume you remove preserve line breaks from the rules for the `Code` element. On import, FrameMaker will translate as spaces the space character entity reference/RE pairs, and as spaces any RE characters (line breaks) not removed by the parser (default behavior). Thus the forced returns (shift-return) are lost and the imported file is not the same as the exported file.
- When importing markup, `preserve line breaks` overrides the `line break is space` rule, if that rule is set. On import, `preserve line breaks` has the same effect for the specified element as the `line break is forced return` rule.

Examples

The following rule preserves line breaks on import and export for the element named `code`:

```
fm element "code" {
is fm element "Code";
preserve line breaks;
}
```

See also

Rules mentioned in synopses	"element" on page 56
Related rules	"line break" on page 142

processing instruction

Use the `processing instruction` rule to drop processing instructions that are not recognized by FrameMaker. By default, the software stores such processing instructions as the marker text in non-element markers of type `DOC PI and DOC Comment`.

Synopsis and contexts

```
processing instruction drop;
```

ArgumentsNone

Details

- When you translate a markup document to FrameMaker and the software encounters an unrecognized processing instruction such as:

  ```
  <?mypi?>
  ```

  it stores the processing instruction as the text of a non-element `DOC PI` marker by default, with the following as the marker text:

  ```
  mypi
  ```

  When you translate a FrameMaker document to markup, it outputs the corresponding processing instruction if it finds a non-element `DOC PI` marker with text in that format.

- This rule does not affect how FrameMaker treats the processing instructions it does recognize for books, book components, and other non-element markers.

Examples

To discard all unrecognized processing instructions, use this rule:

```
processing instruction drop;
```

See also

Rules mentioned in synopses	"drop" on page 53
General information on this topic	Developer Guide, Chapter 21, Translating Entities and Processing Instructions

proportional width resolution is

Use the `proportional width resolution is` rule to change the number used as the total for proportional column widths in tables. By default, if FrameMaker writes proportional columns widths, those widths add to 100.

Synopsis and contexts

```
writer proportional width resolution is "value";
```

Arguments

value	An integer indicating the total for proportional column width values.

Details

Using this rule does not indicate that FrameMaker uses proportional widths, only that if FrameMaker writes proportional widths, then those widths add to *value* instead of 100. To tell FrameMaker to use proportional widths, you must include the `use proportional widths` rule.

Examples

- Assume you do not use the `proportional width resolution is` rule, but have this rule:

  ```
  writer use proportional widths;
  ```

  Further assume you have a 5-column table whose first two columns are 1 inch wide and whose last three columns are 2 inches wide. If the column widths are written to the `colwidth` attribute of the markup `table` element, then FrameMaker creates this start-tag for that table:

  ```
  <table colwidth="12.5* 12.5* 25* 25* 25*">
  ```

- Assume you have the same table as in the last example and you use this rule:

  ```
  writer {
     use proportional widths;
     proportional width resolution is "8";
  }
  ```

  FrameMaker writes this start-tag for the table:

  ```
  <table colwidth="1* 1* 2* 2* 2*">
  ```

- Assume you have the same table as in the previous examples and you use this rule:

  ```
  writer proportional width resolution is "8";
  ```

  That is, you do not also have the `use proportional widths` rule. In this case, FrameMaker ignores the “proportional width resolution” rule and writes this start-tag for the table:

  ```
  <table colwidth="1in 1in 2in 2in 2in">
  ```

See also

Related rules	"use proportional widths" on page 164
General information on this topic	Developer Guide, Chapter 22, Translating Tables

put element

See "generate book" on page 93.

reader

The `reader` rule indicates a rule that applies only on import to FrameMaker. It can be used at the highest level to set a default, or within an `element` rule to specify information particular to that element.

Synopsis and contexts

```
1.element "gi" {. . .
      reader {. . .
         subrule;
   . . .} . . .}

2.reader {. . .
      subrule;
   . . .}
```

Arguments

gi	A markup element's name (generic identifier).
subrule	Valid subrules: `character map` changes how FrameMaker translates between individual characters in the markup and FrameMaker character sets. Allowed only at the highest level. `drop content` imports only the element itself, not its contents. Allowed only within an `element` rule. `end vertical straddle` specifies the end of a vertical straddle in a table. Allowed only within an `element` rule for a table cell or row element. `entity` specifies the treatment of an entity in FrameMaker. Allowed only at the highest level.

`generate book` specifies how to identify book components in a markup document. Allowed only at the highest level.

`insert table part element` specifies that FrameMaker should generate a table part (table title, table heading, or table footing) even if there is no content for that part. Allowed only within an `element` rule for a table element.

`line break` changes the treatment of line breaks in the markup instance which are not handled by the parser on import. Allowed at the highest level or within an `element` rule.

`preserve fm element definition` instructs the software not to modify a FrameMaker element definition when updating an existing EDD. Allowed only at the highest level.

`start new row` specifies that this table cell element starts a new row in the table. Allowed only within an `element` rule for a table row element.

`start vertical straddle` specifies the start of a vertical straddle in a table. Allowed only within an `element` rule for a table cell element.

`table ruling style is` specifies the ruling style to apply to all tables. Allowed only at the highest level.

Examples

To change the default ruling style for tables:

```
reader table ruling style is "thick";
```

reformat as plain text

Use the `reformat as plain text` rule in an `entity` rule for an entity you want to translate as a text inset in FrameMaker. This specifies that the software should remove any element structure from the text inset and reformat the text using the format rules of the document into which the text inset is placed. You specify the other choices for formatting text insets with the rules `reformat using target document catalogs` and `retain source document formatting`.

Synopsis and contexts

```
1.entity "ename" {
      is fm text inset “fname”;
      reformat as plain text;
   . . .}
```

```
2.reader entity "ename" {
      is fm text inset "fname";
      reformat as plain text;
   . . .}
```

Arguments

ename	An entity name.

See also

Related rules	"reformat using target document catalogs," next "retain source document formatting" on page 156
Rules mentioned in synopses	"entity" on page 61 "is fm text inset" on page 136
General information on this topic	Developer Guide, Chapter 21, Translating Entities and Processing Instructions

reformat using target document catalogs

Use the `reformat using target document catalogs` rule in an `entity` rule for an entity you want to translate as a text inset in FrameMaker. This specifies that the software should retain any element structure from the text inset and reformat the text using the format rules of the document into which the text inset is placed. This is the default behavior for entities treated as text insets. You specify the other choices for formatting text insets with the rules `reformat as plain text` and `retain source document formatting`.

Synopsis and contexts

```
1.entity "ename" {
      is fm text inset "fname";
      reformat using target document catalogs;
   . . .}
2.reader entity "ename" {
      is fm text inset "fname";
      reformat using target document catalogs;
   . . .}
```

Arguments

ename	An entity name.

See also

Related rules	"reformat as plain text," (the previous section) "retain source document formatting" on page 156
Rules mentioned in synopses	"entity" on page 61 "is fm text inset" on page 136
General information on this topic	Developer Guide, Chapter 21, Translating Entities and Processing Instructions

retain source document formatting

Use the `retain source document formatting` rule in an `entity` rule for an entity you want to translate as a text inset in FrameMaker. This specifies that the software should remove any element structure from the text inset, but keep the formatting of the source document, rather than reformatting it according to the rules of the document that contains the text inset. You specify the other choices for formatting text insets with the rules `reformat as plain text` and `reformat using target document catalogs`.

Synopsis and contexts

```
1.entity "ename" {
      is fm text inset "fname";
      retain source document formatting;
   . . .}
2.reader entity "ename" {
      is fm text inset "fname";
      retain source document formatting;
   . . .}
```

Arguments

ename	An entity name.

See also

Related rules	"reformat as plain text" on page 154 "reformat using target document catalogs," (the previous section)
Rules mentioned in synopses	"entity" on page 61 "is fm text inset" on page 136
General information on this topic	Developer Guide, Chapter 21, Translating Entities and Processing Instructions

specify size in

Use the `specify size in` rule only in an `element` rule for a graphic or equation element, to provide information the software needs when writing a document containing graphics and equations to markup. This rule determines which of the `dpi` or the `impsize` attribute FrameMaker uses to indicate the size of a graphic or equation. The rule also indicates what units are used for `impsize` and the resolution in which sizes are reported is always 0.001. If there is no `specify size in` rule, FrameMaker uses the `dpi` attribute.

Synopsis and contexts

```
1.element "gi" {
      is fm equation element ["fmtag"];
      writer equation specify size in units
   . . .}

2.element "gi" {
      is fm graphic element ["fmtag"];
      writer anchored frame specify size in units
   . . .}

3.element "gi" {
      is fm graphic element ["fmtag"];
      writer facet "facetname" specify size in units
   . . .}
```

Arguments

gi

A markup element's name (generic identifier).

fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

facetname

A facet name. The string for the facetname must exactly match the string for the facetname in the FrameMaker document. To determine a graphic file's facetname, select the graphic, click Graphics>ObjectProperties, and observe the facetname in the dialog box.

units

The units in which the size of the element is coded. Valid values: `cm`, `cc`, `dd`, `in`, `mm`, `pc`, `pi`, or `pt`.

Details

- Use this rule when you export FrameMaker documents to markup documents.
- FrameMaker reports the size of the elements in the indicated units, at a fixed resolution of 0.001.

Examples

- Suppose your document has a graphic element, `graph`, containing an Anchored Frame sized to fit a FrameMaker-drawn circle with a diameter of 3.15 centimeters. Given the rule:

```
element "graph" {
   is fm graphic element;
   writer anchored frame specify size in cm;
}
```

  FrameMaker generates the attribute `height="3.150cm" and attribute width="3.150cm"`.

- With the same graphic, if the rule is:

```
element "graph" {
   is fm graphic element;
   writer anchored frame specify size in mm;
}
```

  FrameMaker generates `height="31.500mm" and attribute width="31.500mm"`.

See also

Related rules	"convert referenced graphics" on page 51 "entity name is" on page 63 "export to file" on page 69 "specify size in" on page 157
Rules mentioned in synopses	"element" on page 56 "is fm graphic element" on page 114 "is fm equation element" on page 111 "anchored frame" on page 43 "equation" on page 65 "facet" on page 74 "writer" on page 168
General information on this topic	Developer Guide, Chapter 23, Translating Graphics and Equations

start new row

Use the `start new row` rule in the `element` rule for a table cell element to specify that an occurrence of the table cell element indicates that FrameMaker should start a new table row to contain that cell.

Synopsis and contexts

```
element "gi" {. . .
   is fm table cell element ["fmtag"];
   reader start new row ["name"];
. . .}
```

Arguments

gi — A markup element's name (generic identifier).

fmtag

A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

name — An optional name to identify this row

Details

- Your DTD may contain elements that you want to format as tables in FrameMaker even though the element hierarchy does not match that required by FrameMaker for tables. In such a situation, the nature of the element hierarchy may indicate where new rows should begin.
- In some cases, you can use a rule such as the following to indicate that a table cell starts a new row:

  ```
  element "gi" {
     is fm table cell element;
     fm property column number value is "1";
  }
  ```

  With this rule, when FrameMaker encounters a *gi* element, it tries to place that element in the first column of the current table row. If there is already a cell in the first column of the current row, the software automatically creates a new row for *gi*. In this situation, you would not also need the `start new row` rule.

  However, if there is not already a cell in the first column of the current row when the software encounters a *gi* element, it puts the *gi* cell in the current row and does not create a new row for it. This can happen if the table has a vertical straddle in the first column. When FrameMaker encounters a *gi* element on a row that should have a vertical straddle in the first column, with only the rule above, the software puts the *gi* element in the same row (because that cell isn't

occupied). To guarantee a new row starts with the occurrence of *gi* instead, you should use this rule:

```
element "gi" {
   is fm table cell element;
   fm property column number value is "1";
   reader start new row;
}
```

Examples

For a complete example using the `start new row` rule, see Developer Guide, page 354: Omitting explicit representation of table parts.

See also

Related rules	"start vertical straddle," next
Rules mentioned in synopses	"element" on page 56 "is fm table part element" on page 135 "reader" on page 153
General information on this topic	Developer Guide, Chapter 22, Translating Tables

start vertical straddle

Use the `start vertical straddle` rule inside the `element` rule for a table cell to specify that an occurrence of the cell element indicates the start of a vertical straddle.

Synopsis and contexts

```
element "gi" {. . .
   is fm table cell element ["fmtag"];
   reader start vertical straddle "name";
. . .}
```

Arguments

gi	A markup element's name (generic identifier). *fmtag* A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.
name	A name associated with a table straddle. This name must occur in at least one corresponding `end vertical straddle` rule.

Details

- Your DTD may contain elements that you want to format as tables in FrameMaker even though the element hierarchy does not match that required by FrameMaker for tables. In such a situation, the nature of the element hierarchy may indicate where vertical straddles should begin and end. The `start vertical straddle` rule allows you to specify such elements.
- Use this rule in conjunction with the `end vertical straddle` rule. That rule specifies a table cell or row that indicates the end of the vertical straddle started by this rule.
- You give a name to the particular straddle started by *`gi`*. In the corresponding `end vertical straddle` rule or rules, you use the same name to specify that the element ends this vertical straddle.

Examples

For an example of the use of this rule, see Developer Guide, page 357: Creating parts of a table even when those parts have no content.

See also

Related rules	"start new row," (the previous section)
Rules mentioned in synopses	"element" on page 56 "is fm table part element" on page 135 "reader" on page 153
General information on this topic	Developer Guide, Chapter 22, Translating Tables

table ruling style is

You use the `table ruling style is` rule to specify the ruling style for all tables.

Synopsis and contexts

```
reader table ruling style is "style";
```

Arguments

`style`	A ruling style for all tables. One of the keywords: None, Double, Medium, Thick, Thin, or Very Thin.

Details

- This rule specifies the ruling style applied to all tables. When working with the CALS table model, you can use the `frame`, `colsep`, and `rowsep` attributes to determine whether or not portions of a table have rulings. However, these attributes have boolean values. Consequently, you can only use them to say whether or not a table has a ruling, not what type of ruling to

use if it does have one. In this situation, you could use the `table ruling style is` rule to set the ruling style for all tables.

- FrameMaker considers the ruling style set with this rule as custom ruling. If you re-import formats to the FrameMaker document and remove overrides, the ruling style set with this rule will remain. If possible, therefore, you should use table formats to specify ruling styles.

Examples

To specify that all tables should use the Thick ruling style, use this rule:

```
reader table ruling style is "Thick";
```

See also

General information on this topic — Developer Guide, Chapter 22, Translating Tables

unwrap

Use the `unwrap` rule when you do not want to preserve an element on translation from one representation to another. If you specify that FrameMaker should unwrap an element (*`gi`* or *`fmtag`*), the software places the element's content as part of the content of the element's parent element, but does not make an element for *`gi`* or *`fmtag`* itself.

Synopsis and contexts

```
1.element "gi" unwrap;
2.fm element "fmtag" unwrap;
```

Arguments

`gi` — A markup element's name (generic identifier).

`fmtag` — A FrameMaker element tag. These names are case-sensitive and must appear in the rule the same as in the EDD.

Details

- When FrameMaker encounters an element to be unwrapped, it does not insert a corresponding element into the document it is creating. Instead, it inserts the content of an unwrapped element.
- If you use this rule to unwrap FrameMaker cross-reference elements or system variable elements, those elements become text in the resulting markup document.
- When importing a DTD or exporting an EDD, FrameMaker does not generate an element definition or declaration corresponding to an element that is unwrapped. Furthermore, when

an element uses the unwrapped element in its definition, the software replaces the name of the unwrapped element with its content model or general rule in the general rule or content model of the element that used it or replaces it with the list of its children in an exception. You can change this behavior by using the `preserve fm element definition` rule.

- You cannot use the `unwrap` rule with any other subrule of the `element` or `fm element` rules. For example, you cannot specify that a markup element both be unwrapped and be translated to a FrameMaker element.

Examples

- A markup document used to produce both the student's and teacher's edition of a textbook might include an `ANSWER` element used for answers to exercises. In producing the teacher's edition of the textbook, this element might be unwrapped into FrameMaker as text. A structure API client could associate this element with the condition tag `Answer`.
- Suppose a DTD contains the following declarations:

```
<!ELEMENT wrapper - - (a, b)>
<!ELEMENT x - - (p, q, wrapper, r)>
<!ELEMENT y - - (#PCDATA) +(wrapper)>
```

and you have this rule:

```
element "wrapper" unwrap;
```

FrameMaker would generate the following element definitions:

Element (Container): X
General rule: P, Q, A, B, R

Element (Container): Y
General rule: <TEXT>
Inclusions: A, B

See also

Related rules	"preserve fm element definition" on page 148 "drop" on page 53
Rules mentioned in synopses	"element" on page 56 "fm element" on page 77
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes

use processing instructions

See "generate book" on page 93.

use proportional widths

Use the `use proportional widths` rule to indicate that when FrameMaker writes the width of table columns, it should use proportional measurements. By default, if the software writes the width of table columns, it uses absolute measurements.

Synopsis and contexts

```
writer use proportional widths;
```

ArgumentsNone.

Details

- If you use this rule when writing an attribute indicating the width of one or more columns in a table, FrameMaker writes values such as `"25*"`, where the asterisk `*` indicates a proportional measurement, instead of values such as `"0.25in"` which are absolute measurements.
- If you use this rule, you can also use the `proportional width resolution is` rule to determine what number the values add to. Without the `proportional width resolution is` rule, the proportional measurements add to 100.

Examples

- Assume you do not use the `proportional width resolution is` rule, but have this rule:

  ```
  writer use proportional widths;
  ```

  Further assume you have a 5-column table whose first two columns are 1 inch wide and whose last three columns are 2 inches wide. If the column widths are written to the `colwidth` attribute of the markup `table` element, then FrameMaker creates this start-tag for that table:

  ```
  <table colwidth="12.5* 12.5* 25* 25* 25*">
  ```

- Assume you have the same table as in the last example and you use this rule:

  ```
  writer {
     use proportional widths;
     proportional width resolution is "8";
  }
  ```

  FrameMaker writes this start-tag for the table:

  ```
  <table colwidth="1* 1* 2* 2* 2*">
  ```

See also

Related rules	"proportional width resolution is" on page 152
General information on this topic	Developer Guide, Chapter 22, Translating Tables

value

Use the `value` rule to translate the value of a markup attribute to the value of a particular FrameMaker property or to a particular choice for a FrameMaker choice attribute. The attribute's declared value must be a name token group or `NOTATION` and a name token group.

Synopsis and contexts

```
1.value "token" subrule;

2.attribute "attr" {. . .
      value "token" subrule;
      . . .}

3.element "gi" {. . .
      attribute "attr" {. . .
         value "token" subrule;
         . . .} . . .}
```

Arguments

token	A token in a name token group.
attr	The name of a markup attribute.
gi	A markup element's name (generic identifier).
subrule	One of the following: `is fm value` translates a markup value to a particular choice for a FrameMaker choice attribute. `is fm property value` translates a markup value to the value of a particular FrameMaker property.

Details

The rule can be used at the highest level to set a default, within a highest-level attribute rule to set the default for all attributes that use that token, or within an `element` rule to set the default for a particular token within a particular attribute in that element.

Examples

- To rename the FrameMaker `import by reference or copy` property as the `refcopy` attribute, and to also change the name tokens, use this rule:

```
attribute "refcopy" {
   is fm property import by reference or copy;
   value "r" is fm property value reference;
   value "c" is fm property value copy;
}
```

- If the token list `(r | b | g)` is used by multiple attributes, you can use these rules to translate the individual tokens consistently:

```
value "r" is fm value "Red";
value "b" is fm value "Blue";
value "g" is fm value "Green";
```

- If the token list `(r | b | g)` is used by several attributes as above, but by the `bird` element differently, you can add this rule to the above rules:

```
element "bird" {is fm element;
] attribute "species" {
   value "r" is fm value "Robin";
   value "b" is fm value "Blue Jay";
   value "g" is fm value "Goldfinch";
   }}]
```

See also

Related rules	"is fm value" on page 139 "is fm element" on page 110
Rules mentioned in synopses	"attribute" on page 46 "element" on page 56
General information on this topic	Developer Guide, Chapter 20, Translating Elements and Their Attributes

value is

See "fm property" on page 80.

write structured document

By default, when you save a FrameMaker document to markup, the software writes out the document instance, any declarations for the internal DTD subset, and a `DOCTYPE` statement which references the external DTD subset, but (for SGML) not an SGML declaration nor the declarations within the external DTD subset. If an XML structure application (in `structapp.fm`) specifies a Schema file for output, that file is also written with the XML document. You can use this rule to confirm the default behavior.

Synopsis and contexts

```
writer write structured document;
```

ArgumentsNone.

Details

You cannot use the `write structure document` rule and the `write sgml document instance only` rule in the same read/write rules file.

See also

Related rules

"external dtd" on page 73
"include dtd" on page 98
"include sgml declaration" on page 100
"write structured document instance only," next

write structured document instance only

By default, when you save a FrameMaker document to markup, the software writes out the document instance, any declarations for the internal DTD subset, and a `DOCTYPE` statement which references a file for the external DTD subset. For SGML, it does not write an SGML declaration. This rule causes the software to write the document instance only--no external or internal DTD, no Schema, and no SGML declarations.

Synopsis and contexts

```
writer write structured document instance only;
```

ArgumentsNone.

Details

- By default, when you translate a FrameMaker document to markup, as its last step the software runs the parser on the markup document to check its validity. If you use this rule, FrameMaker does not write a complete markup document and so does not send the result through the parser.
- You cannot use the `write structure document instance only` rule in the same read/write rules file as any of the `write structure document`, `include dtd`, or `include sgml declaration` rules.

See also

Related rules

"external dtd" on page 73
"include dtd" on page 98
"include sgml declaration" on page 100
"write structured document," (the previous section)

writer

The `writer` rule indicates a rule that applies only on export of a FrameMaker document to markup. It can be used at the highest level to set a default or within an `element` rule to specify a subrule for that element.

Synopsis and contexts

```
1.writer {. . .
      subrule;
   . . .}

2.element "gi" {. . .
      writer {. . .
         subrule;
   . . .} . . .}
```

Arguments

gi — A markup element's name (generic identifier).

subrule — Valid subrules:

`anchored frame` tells FrameMaker what to do with graphic elements other than those with a single non-internal FrameMaker facet. Allowed only within an `element` rule for a graphic element.

`character map` determines the correspondence between individual characters in the FrameMaker and markup character sets. Allowed only at the highest level.

`convert referenced graphics` tells the software to create new files for graphic files that were imported by reference. `drop content` exports a FrameMaker element without its contents. Allowed only within an `element` rule.

`equation` tells FrameMaker what to do with equation elements. Allowed only with an `element` rule for an equation element.

`external dtd` specifies an external DTD to use. Allowed only at the highest level.

`facet` tells FrameMaker what to do with a graphic element that has a single non-internal FrameMaker facet. Allowed only with an `element` rule for a graphic element.

`[do not] include dtd` specifies information to exclude or include in the written document. Allowed only at the highest level.

`[do not] include sgml declaration` specifies information to exclude or include in the written document. Allowed only at the highest level.

`line break` specifies treatment of line breaks not handled by the parser on export. Allowed at the highest level or within an `element` rule.

`[do not] output book processing instructions` specifies whether or not to create processing instructions that identify book components when writing a FrameMaker book as a markup document. Allowed only at the highest level.

`proportional width resolution is` specifies the total value to which proportional widths for table columns add up. Allowed only at the highest level.

`use proportional widths` specifies that the software should use proportional values in describing the widths of table columns. Allowed only at the highest level.

`write structured document` specifies that an entire SGML document should be written, not just the document instance. This is the default. Note that the external DTD subset is not written to the file. Instead, a DOCTYPE statement with a reference to the external DTD file is written. Allowed only at the highest level.

`write structured document instance only` specifies that only the document instance should be written, not the DTD and SGML declaration. Allowed only at the highest level.

Examples

- To tell FrameMaker not to use processing instructions to identify book components when writing a FrameMaker book as a markup document, use this rule:

```
writer do not output book processing instructions;
```

- Assume you want all graphics to be exported in TIFF format. Further assume that some of your graphic elements were imported from the TIFF format. For these elements you don't want to create a new external data entity. To accomplish this, use these rules:

```
element "graphic" {
   is fm graphic element;
 writer facet default{
 convert referenced graphics;
 export to file "$(entity) .tif as "TIFF";
   writer anchored frame
      export to file "$(entity).tif" as "TIFF";
}
```

4 Conversion Tables for Adding Structure to Documents

You can set up a conversion table to help end users automate the task of adding structure to documents. The conversion table uses paragraph and character formats to identify which unstructured document objects to wrap in elements, and element tags to identify which child elements to wrap in parent elements. A user wraps all of a document's contents in one move by applying a structure command to the document and referring to one of your conversion tables.

This chapter describes how to set up a conversion table and define object and element mapping in it. For information on the commands for adding structure to documents, see the FrameMaker user's manual

How a conversion table works

A conversion table contains rules for mapping between document objects and elements and between child elements and parent elements. The table is a regular FrameMaker table, with at least three columns and one body row. Each body row holds one rule.

The first column in a conversion table specifies a document object, a child element, or a sequence of child elements or paragraphs to wrap in an element. A *document object* is a paragraph, text range, table, table part (such as heading or row), equation, variable, footnote, Rubi group, Rubi text, marker, cross-reference, text inset, or graphic (anchored frame or imported graphic object).

The second column in the table specifies the element in which you want to wrap the object, child element, or sequence. The third column can specify an optional *qualifier* to use as a temporary label for the element in rules that are applied later. For example:

Wrap this object	In this element	With this qualifier
P:BulletItem	Item	Bullet
E:Item[Bullet]+	BulletList	

The first column uses a one-letter code and usually a tag to identify an object or element.

The second column specifies the element in which to wrap the object or element.

The third column can provide a label for the new element to be used in later rules.

To add structure to a document or book, an end user chooses the **Structure Current Document...**, **Structure Documents...**, or **Structure Current Book...** command from the **StructureTools > Utilities** submenu and refers to one of the conversion tables.

When you add structure to a document manually, you typically begin with the lowest-level components and work up to the highest level. For example, to add structure to a chapter you might start by wrapping sub-paragraph objects like text ranges and tables, then wrap the contents of paragraphs together in `Paragraph` elements, then wrap sequences of `Head` and `Paragraph` elements in `Section` elements, and so on until the entire document is wrapped in a single highest-level `Chapter` element.

The process of adding structure with a conversion table is similar to adding structure manually. FrameMaker begins by applying rules to document objects below the paragraph level, then applies rules at the paragraph level, and proceeds through successively higher levels. The process stops when FrameMaker reaches a single highest-level element or when no more rules can be applied. To understand this process, it helps to have manually structured a document.

Using the sample table above, FrameMaker first wraps each paragraph with the paragraph format `BulletItem` in an element called `Item` and gives the element a qualifier called `Bullet`. Then it wraps each `Item` element with the qualifier `Bullet` in a parent element called `BulletList`.

FrameMaker tries to order the rules as much as possible. If a rule needs a building block that is generated by a later rule, the later rule is run first so that all of the building blocks in the first rule are available. To make a conversion table easy to interpret for a human reader, you may want to write the rules in the order they should be applied.

Setting up a conversion table

You can have FrameMaker generate an initial conversion table for you from an unstructured document or book, or you can create a conversion table entirely from scratch. If you already have a document that end users need to add structure to, or a document that is similar to one users will add structure to, you'll probably want to let FrameMaker generate the initial table. You can modify the rules in the table as necessary.

After creating a conversion table, you can update it from other unstructured documents. Updating a table adds rules for any objects in the document that are not yet in the table.

A conversion table document can include the conversion table itself (which may be split up into several tables) and text or graphics you want to include for documenting the rules. It cannot have any tables other than conversion tables. You need to save the document before it can be used for adding structure to other documents or books.

Each body row in a conversion table holds one mapping rule. FrameMaker reads only the information in the first thee columns of the body rows, so you can use additional columns and headings and footings for comments about rules.

For information on defining and modifying the rules in a table, see "Adding or modifying rules in a conversion table" on page 175.

Generating an initial conversion table

You can have FrameMaker generate a conversion table from an unstructured document. This is the easiest way to begin a new conversion table.

To generate an initial conversion table, choose **Generate Conversion Table** from the **StructureTools** menu in a document with objects you want to structure. Select Generate New Conversion Table in the dialog box and click Generate.

The software looks through the flows on body pages in the document and compiles a list of every object that can be structured. For each object, it gives the object type and the format tag used in the document (if the object has a format), and maps the object to an element. The element tag is the same as the format tag, or if the object does not have a format, the element tag is a default name such as `CELL` or `BODY`. If necessary, FrameMaker removes parentheses and other characters to create an element tag that is valid.

The initial conversion table gives you a first pass through the document, identifying objects to wrap in elements. It does not identify child elements to wrap in parent elements—you need to add those rules to the table yourself.

This is an example of an initial conversion table:

Wrap this object	In this element	With this qualifier
P:Head1	Head1	
P:Head2	Head2	
P:Body	Body	
P:Code	Code	
SV:Current Date \(Long\)	CurrentDateLong	
C:Code	cCode	
TC:	CELL	
TR:	ROW	

For details on the object type identifiers used in the table (such as `P:` and `TC:`), see "Identifying a document object to wrap" on page 177.

Note that if there are conflicts in a format tag from the unstructured document, an object type identifier in lowercase is prepended to any duplicate element tag. In the example above, the

element tag for text ranges with the `Code` character format is `cCode` because the document also has a paragraph format called `Code`.

When you create an initial table, FrameMaker does not examine the document's format catalogs—it looks only at objects actually used in the document. For this reason, the table may not be as complete as you need. You may want to update the table from a set of documents that together provide all or most of the objects you need rules for. You can also add and modify rules manually.

The initial convertion table does not contain a root element for the structure hierarchy, but you can add one manually, using the tag `RE:RootElement`, so that documents you convert using the table will have a "well formed" structure in which all elements are children of the root element. See "Specifying the root element for a structured document" on page 176.

The initial conversion table does contain elements for all defined paragraph and character formats that are used in the unstructured document, and for all objects, including cross references, markers, footnotes, equations, graphics, system variables, and tables. Formatting is retained in the structured document created from the table, and carried forward into the EDD in *ParagraphFormattingTag* elements.

If the original document contains format overrides or unnamed formats applied directly to text, you can create named formats from them before conversion, or flag them for manual update in the conversion table. See "Flagging format overrides" on page 184 and "Wrapping untagged formatted text" on page 184.

Setting up a conversion table from scratch

You can set up a regular FrameMaker table to serve as a conversion table. The table must appear on a body page in its own document. The document and table can be structured or unstructured. Begin a conversion table this way if you do not yet have an unstructured document to use for generating the table.

To set up a conversion table from scratch, create a new document and insert a table with at least three columns and one body row. The table can have any number of heading or footing rows.

You can divide a conversion table into several smaller tables. This is helpful when you have many rules and want to organize the rules in groups. Each table must have at least three columns and one body row. You can add explanatory heads and paragraphs between the tables to document the rules. Do not include tables that are not conversion tables.

Updating a conversion table

After creating a conversion table, you may want to update the table from at least one other unstructured document to get a more complete list of objects. FrameMaker adds a rule for each object from the document that is not already listed in the table.

To update a conversion table, choose **Generate Conversion Table...** from the **StructureTools** menu in a document with the objects you want to structure. Select the name of the conversion table document in the **Update Conversion Table** popup menu and click **Generate**.

When you update a conversion table, the process that FrameMaker goes through is similar to the process of generating an initial table. The software does not examine the document's format catalogs—it looks only at objects actually used in the document.

Adding or modifying rules in a conversion table

Each body row in a conversion table holds one mapping rule. Follow these steps to define a mapping rule:

1. **In the first column, identify a document object, a child element, or a sequence of child elements or paragraphs to wrap.**

   You use a one- or two-letter code to identify the type of item and, in most cases, a format or element tag to narrow the definition. See "Identifying a document object to wrap" on page 177, "Identifying an element to wrap" on page 178, or "Identifying a sequence to wrap" on page 179.

2. **In the second column, specify an element in which to wrap the object, child element, or sequence.**

   Type one valid element tag. If you are writing rules for a document that already has element definitions, use tags from the document's Element Catalog.

   If you are wrapping a table part, graphic, or inset, FrameMaker always wraps all instances of the object in the same kind of element. The element has a default tag, such as `CELL`, `BODY`, `GRAPHIC`, or `INSET`. Type a different tag in the second column only if you want to override the default tag.

   You can also give an element an attribute with a value. For details, see "Providing an attribute for an element" on page 181.

3. **(Optional) In the third column, add a qualifier for the new element tag.**

   A qualifier is a temporary label that you can attach to an element tag for the structuring process. If you wrap the element in a parent element in a later rule, you include the qualifier tag with the element tag. For details, see "Using a qualifier with an element" on page 181.

To make a conversion table easy to read and to help you think through the process, we recommend that you put the rules in order from the lowest level to the highest. In the first rows of the table, write rules that wrap individual document objects such as text ranges, tables, and paragraphs; next add rules that wrap child elements in parent elements; then add rules that wrap sequences in elements; and finally add rules that wrap elements in one root element.

Every flow in a document must have a highest-level element, and the element can be different for each flow.

About tags in a conversion table

Format and element tags in a conversion table are case-sensitive and must be specified the way they are defined in their catalogs. Qualifier tags are also case-sensitive, and two occurrences of

one qualifier must match exactly. The following characters are not allowed in an element tag, but can appear in a format or qualifier tag if you precede them with a backslash (\) in the table:

```
( ) & | , * + ? % [ ] : \
```

A space character does not need to be preceded with a backslash. For example, you can write the tag `Format A`.

You can use a percentage sign (%) as a wildcard character in a format or element tag to match zero, one, or more characters. For example, `P:%Body` matches paragraphs with the format tag `Body`, `FirstBody`, or `BulletBody`.

Specifying the root element for a structured document

FrameMaker now allows you to specify a *root element*, the highest valid element in a document, so that the converted document adheres to structured document convention.

To do so, specify the optional `RE:RootElement` after conversion. You must add it manually to the conversion table, specifying the tag itself, `RE:RootElement`, in the first column, and the element name that you choose in the second column.

When you generate a structured document using this manually modified conversion table, the resulting document contains a well-formed hierarchy with a valid root element. If you convert an entire book using the table, each document contains a valid root element.

The root element name that you choose should be unique within the document. If you specify a name that its being already defined for some other object, the root element is ignored. You can still generate a structured document with the table, but it will not have a valid root element, and a message is added to the FrameMaker Log window: “Element name defined in second column of conversion table for root element is not unique. Root element ignored.”

The root element tag should appear only once in the conversion table. If it appears anywhere else with a different name, it is ignored and a generated document does get a root element, but if it appears twice with the same name, both elements are ignored and a generated document will have no root element.

If no root element is generated for a document (either because the conversion table contains no `RE:RootElement` tag or because it is not specified correctly), the ‘NoName’ element appears at the top of the element hierarchy. The rest of the elements are its children, and the hierarchy is shown to have an invalid structure.

The `RE:RootElement` is particularly useful for unstructured documents that do not easily conform to the required structure rules, maybe due to poor adherance to tagging rules or too many manual style overrides. In these cases it may be uneconomic to tailor your conversion table for every possible formatting variation.

Identifying a document object to wrap

To identify a document object to wrap in an element, type an object type identifier and (optionally) a format tag in the first column of the table. Separate the identifier and format tag with a colon.

FrameMaker finds all the objects with that type and format and wraps them in the element you specify in the second column of the table. If you leave the format tag out of the rule, FrameMaker finds all the objects with the specified type that are not identified in other conversion rules.

For example:

Wrap this object	In this element
P:Body	Para
T:RulesTbl	RulesTbl
T:	StandardTbl
Q:Small	SmallEqns

This rule wraps all tables not named in other rules, regardless of format tag.

These are the object type identifiers and format tags you can use:

Object type	Identifier	Format tag
Paragraph	`P:`	Paragraph format tag
Text range	`C:`	Character format tag
Table	`T:`	Table format tag
Table title	`TT:`	(none)
Table heading	`TH:`	(none)
Table body	`TB:`	(none)
Table row	`TR:`	(none)
Table cell	`TC:`	(none)
System variable	`SV:`	Variable format name
User variable	`UV:`	Variable format name
Graphic (anchored frame or imported object)	`G:`	(none)
Footnote	`F:`	Location of footnote: `Table` or `Flow`
Rubi group	`RG:`	(none)
Rubi text	`R:`	(none)
Marker	`M:`	Marker type

Object type	Identifier	Format tag
Cross-reference	X:	Cross-reference format name
Text Inset	TI:	(none)
Equation	Q:	Size of equation: Small, Medium, or Large

Table parts, graphics, and text insets do not have any formatting information, so FrameMaker wraps all instances of those objects in the same kind of element. The element has a default tag, such as CELL, BODY, GRAPHIC, or INSET. (Specify a different tag in the second column to override the default tag.)

You can write identifiers and the keywords for footnote location or equation size in any combination of uppercase and lowercase letters. The names of formats and marker types are case-sensitive, however, and must be typed the way they are specified in their catalogs.

A system variable can be wrapped in a variable element but a user variable cannot. If you identify a user variable, FrameMaker wraps it in a container element with the tag specified in the second column.

FrameMaker wraps a text inset in a container.

Identifying an element to wrap

To identify a child element to wrap in a parent element, type the object type identifier E: followed by an element tag and (optionally) a qualifier in brackets in the first column of the table. The qualifier must already be defined for the element in a rule applied earlier.

FrameMaker finds all instances of the element and wraps them in the element you specify in the second column of the table. You can omit the element tag if you include a qualifier.

For example:

Wrap this object	In this element
E:Item[Bullet]	BulletItem
E:[1Head]	ChapHead

This rule wraps all elements with the qualifier 1Head not named in other rules.

You can type the E: identifier in either uppercase or lowercase. The element tags are case-sensitive, however, and must be typed the way they are specified in their catalog. You can even omit the E: identifier—when FrameMaker reads an object name with no identifier, it assumes the object is an element.

To identify a table child element to wrap in a table parent element, type the object identifier TE: followed by E:, an element tag, and (optionally) a qualifier in brackets in the first column of the

table. This allows you to name a table element from one or more child elements, rather than naming it from a table format tag (with the `T:` identifier).

For example:

Wrap this object	In this element
TB:RulesBody	RulesBody
TE:E:RulesBody	RulesTbl

This rule wraps RulesBody table child elements in a RulesTbl table element.

Most often, you wrap multiple elements together in one parent. You can use `E:` or `TE:` to identify a sequence of elements for this. For more information, see "Identifying a sequence to wrap," next. For more information on qualifiers, see "Using a qualifier with an element" on page 181.

Identifying a sequence to wrap

You can wrap a sequence of child elements in a parent element. For example, you might wrap a `Head` element followed by one or more `Paragraph` and `List` elements in a higher-level `Section`.

You can also wrap a sequence of unwrapped paragraphs in an element. For example, you might wrap a sequence of paragraphs with the format tag `Body` all in one `Note` element. (With other unwrapped document objects such as tables, graphics, and text ranges, you can wrap only one object in an element.)

To identify a sequence to wrap, specify object type identifiers and element tags or paragraph format tags, and use symbols to further describe the sequence. You can mix elements and unwrapped paragraphs together in one specification.

These are the symbols you can use:

Symbol	Meaning
Plus sign (`+`)	Item is required and can occur more than once.
Question mark (`?`)	Item is optional and can occur once.
Asterisk (`*`)	(SGML only) Item is optional and can occur more than once.
Comma (`,`)	Items must occur in the order given.
Ampersand (`&`)	Items can occur in any order.
Vertical bar (`|`)	Any one of the items in the sequence can occur.
Parentheses	Beginning and end of a sequence.

The symbols available are the same connectors, occurrence indicators, and parentheses used in general rules in an EDD.

For example:

To identify this sequence	Use this specification
One or more `Item` elements	`Item+`
An element tagged `Item[Bullet]` followed by one or more unwrapped paragraphs tagged `Bullet`	`E:Item[Bullet], P:Bullet+`
A `ChapNum` element followed by a `ChapName` element	`ChapNum, ChapName`
A `Head` element followed by zero or more `Paragraph`, `BulletList`, or `NumberList` elements	`Head, (Paragraph|BulletList|NumberList)*`
An `Item[FirstNItem]` element followed by one or more `Item[NItem]` elements	`Item[FirstNItem], (Item[NItem])+` or `[FirstNItem], ([NItem])+`
A `RulesTitle` table title element followed by a `RulesBody` table body element	`TE:E:RulesTitle, E:RulesBody`

Strict or loose sequence specification

If you already have a well defined or standard based application structure, you may try to use the general rule specification as it is defined in your EDD. In many cases, with well formatted unstructured documents, you will achieve excellent conversion results. However, in practice unstructured documents often break the rules. You will find incorrect tagging, manual formatting overrides and other non-standard features.

Your strict conversion table will not cope well with these souce documents. It will fail to wrap sequences that do not match a strict specification. You can avoid these problems by providing a less restrictive sequence specification.

The revised sequence specification must be compatible with the required structure for example if the EDD specified this general rule:

Head, Para+, Table?, Graphic?, Section*

The strict sequence specification could be identical. However, if the conversion table encountered a document whith no `Head` element or a `Para` between `Table` and `Section`, the entire sequence will not be wrapped. The revised sequence specification could be:

Head?, (Para | Table| Graphic)*, Section*

This will give the correct conversion when the source document is well tagged but will also cope with a wide range of variations.

Providing an attribute for an element

When you specify an element in the second column of the table, you can provide an attribute for the element. In the structured document, all the element instances will have the attribute name and value.

To provide an attribute for an element, type the attribute name and value in brackets after the element tag in the second column of the table. Separate the name and value with an equal sign, and enclose the value in double quotation marks.

For example:

Wrap this object	In this element
P:Intro	Para[Security="Unclassified"]
P:Important	Note[Label="Important"]
E:Item+	List[Type="Numbered"]

If the unstructured document has an Element Catalog with an element and attribute matching the one you're providing, the attribute is the type specified in the catalog. If the attribute does not match an attribute already defined, the type is string.

If you need to use a double quotation mark in an attribute value, escape the quotation mark with a backslash (\). Other restrictions on characters are determined by the attribute's type. (The string type allows any arbitrary text string.)

To give an element more than one attribute, separate the attribute definitions with an ampersand (&). For example, this specification gives the element a `Type` attribute with the value `Numbered` and a `Content` attribute with the value `Procedure`:

```
List [Type="Numbered" & Content="Procedure"]
```

For an example of an attribute that maintains formatting information from a qualifier, see "Using a qualifier with an element," next.

Using a qualifier with an element

Qualifiers act as temporary labels that preserve formatting information from the unstructured document until all elements have been wrapped. Qualifiers are used only in the conversion table—they do not show up in a final structured document.

To use a qualifier with an element specified in the second column of the table, type the qualifier tag in the third column. Then when you wrap the element in a later rule, type the qualifier tag in brackets after the element tag in the first column. Spell and capitalize the qualifier the same way

in the two places. FrameMaker keeps track of qualifiers separately from elements, so you can use the same tag for an element and its qualifier.

For example:

Wrap this object	In this element	With this qualifier
P:BulletItem	Item	bulleted
P:NumberItem	Item	numbered
E:Item[bulleted]+	BulletList	
E:Item[numbered]+	NumberList	

First specify the qualifier for the element.

Then include the qualifier with the element in later rules.

In the example above, an unstructured document has both bulleted items and numbered items, with paragraph formats called `BulletItem` and `NumberItem`. When adding structure to the document, you want to wrap all the items in an `Item` element with a parent element of either `BulletList` or `NumberList`. To do this, you need to keep the `BulletItem` and `NumberItem` formatting designations long enough to determine in which list to wrap the items. The conversion table first associates qualifiers called `bulleted` and `numbered` with new `Item` elements. Then it wraps each `Item` element in either a `BulletList` or a `NumberList`, as specified by its qualifier.

Note that if you specify an attribute for formatting information in the second column, you cannot use the attribute as a label for preserving formatting during the conversion process. You still need to use the qualifier. For example:

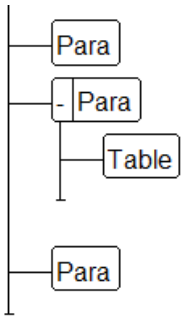
Wrap this object	In this element	With this qualifier
P:BulletItem	Item	bulleted
P:NumberItem	Item	numbered
E:Item[bulleted]+	List[Type="Bulleted"]	
E:Item[numbered]+	List[Type="Numbered"]	

Handling special cases

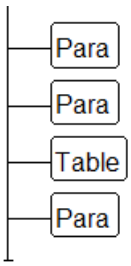
You may need to accommodate a few special circumstances or requirements in a conversion table.

Promoting an anchored object

In an unstructured FrameMaker document, a table or an anchored graphic must be anchored in a paragraph. The anchor specifies which paragraph to keep the object with as an author continues to edit the document. When a user adds structure to the document, the table or graphic normally becomes a child of the paragraph with the anchor, like this:



In a structured document, you often want a table or graphic element to be at the same level as its surrounding paragraph elements. FrameMaker can break the table or graphic out of its paragraph and promote the element to be a sibling of the paragraphs, like this:



To break a table or graphic out of its paragraph and promote it one level, add the keyword `promote` in parentheses after the element tag for the table or graphic. (The keyword is not case-sensitive.) For example:

Wrap this object	In this element
T:Table	Table (promote)

Note that FrameMaker promotes the object at the location of the anchor symbol in the paragraph. If the symbol is in the middle of the paragraph, the structured document will have half of the paragraph, then the table, and then the other half of the paragraph. Typically, you want the symbol to be at the end of the paragraph.

Flagging format overrides

An unstructured document may have *format overrides*. This happens when someone uses the Paragraph or Character Designer to make formatting changes to a paragraph or text range but does not save the changes in the catalog format.

When an end user adds structure to a document, FrameMaker does not normally identify format overrides. You can have FrameMaker flag all element instances in the document that have overrides so that the user can find the overrides and decide how to handle them in a structured context.

To flag format overrides, add the rule `flag paragraph format overrides` or `flag character format overrides` to the first column of the table. (The rule is case-insensitive.) This is a general instruction for the table, so you do not add anything to the second and third columns. For example:

Wrap this object	In this element
flag paragraph format overrides	
flag character format overrides	

At each element instance that has an override in the document, FrameMaker adds an attribute called `Override` with the value `Yes`.

Note: Use the FrameMaker utility "Create and Apply Formats" before conversion to turn format overrides and untagged formatted text into named paragraph and character formats, which can be carried forward automatically into the structured document and EDD.

Wrapping untagged formatted text

It is possible for someone to format a text range by applying commands from the Font, Size, and Style submenus in the Format menu—and not use a character format at all. This leaves the text formatted but without a tag that you can refer to in your conversion table.

You can have FrameMaker find text that has been formatted with the submenu commands and wrap it in a “catch-all” element. After adding structure to a document, the end user will probably

want to look at these instances and change them to other elements (such as `Emphasis`) that more specifically describe the type of formatting.

To wrap untagged formatted text, add the rule `untagged character formatting` to the first column of the table and add an element to the second column. (The rule is case-insensitive.) For example:

Wrap this object	In this element
untagged character formatting	UntaggedText

This might also be useful while you are developing a conversion table. You can add structure to a sample document with this rule to see if the document has any untagged formatting.

Note: Use the FrameMaker utility "Create and Apply Formats" before conversion to turn format overrides and untagged formatted text into named paragraph and character formats, which can be carried forward automatically into the structured document and EDD.

Nesting object elements

Typically, a non-paragraph object such as a table or graphic is wrapped in an object element and then wrapped in a paragraph element. You can also wrap the object in more than one level below the paragraph. Sometimes you need to do this to conform to a DTD that requires more hierarchy, or you may just want to be able to use two objects together.

To nest object elements in a paragraph, define each mapping in a separate rule in the table. For example:

Wrap this object	In this element
M:Index	Index
G:	Graphic
E:Index & E:Graphic	Figure

In the example above, the rules wrap an index marker in an `Index` element and a graphic anchor in a `Graphic` element, and then they wrap the two elements together in a `Figure` text range element. This way, the graphics in a structured document will automatically have a marker identifying a location to be included in an index.

Building table structure from paragraph format tags

When FrameMaker adds structure to tables, it normally wraps all instances of a table part in the same kind of element and uses a default name for the element, such as `CELL`, `ROW`, `HEADING`, or `BODY`. You can override the default name by providing a different element tag in the second column of the conversion table.

If you want to have more than one kind of element for a particular table part, you can build the structure up from the format tags used in the cells or titles. This allows you to distinguish between different formatting used in different instances of a single table part. For example, a table may have a few special body rows with italicized text that marks divisions in the table. Or a table may have two titles, one of them a subtitle in a different font size.

To build table structure from paragraph format tags, for each cell or title rule use the `TC:` or `TT:` type identifier followed by the `P:` identifier and a format tag in the first column of the table. For example:

Wrap this object	**In this element**
TC: P:DividerCell	DividerCell
TC: P:BodyCell	BodyCell
TR:DividerCell+	ROW
TR:BodyCell+	ROW
TB:Row+	BODY

In the example above, the rules map cells that use a `DividerCell` paragraph format in an element called `DividerCell` and map cells that use a `BodyCell` paragraph format in an element called `BodyCell`. Then they wrap both kinds of cell elements in the same default `ROW` element and continue the wrapping normally.

Testing and correcting a conversion table

You should test and correct a conversion table as you develop it. To do this, prepare a sample document that represents the type of documents the table will apply to, and use the conversion table to add structure to the sample. Make sure your sample document has all of the document objects that the final documents may contain.

When a structure command reads a conversion table, it identifies any syntax errors in the rules and displays the errors in a log file. Correct the table and test it again until no more errors are found.

You may find it helpful to wrap only document objects for your first testing pass, without wrapping in higher levels of hierarchy. When you're sure that the rules for wrapping individual

objects are correct, start writing and testing the rules to wrap elements and sequences in parent elements.

5 CSS to EDD Mapping

This chapter provides a reference for the CSS 3 and CSS 2 to EDD mapping feature. The CSS 3 to EDD Mapping section contains the CSS 3 properties that are available for EDD mapping. Similarly, CSS 2 to EDD Mapping section contains the CSS 2 properties that you can map with EDD.

CSS 3 to EDD Mapping

This section provides a reference for the Cascading Style Sheet (CSS) version 3.0 to EDD mapping and importing feature. The topic covers the FrameMaker properties available in various pod and their corresponding CSS property.

Each property's description includes the following headings.

FrameMaker property Name of the property in the properties pod.

CSS property The CSS 3.0 property name

CSS Property Values A simple list of the available property values.

Comments/Values Additional information about the mapping includes EDD element property values.

While importing a CSS into an EDD, any property or selector in the CSS that cannot be mapped to an equivalent EDD rule is ignored by FrameMaker. For example, CSS Specificity does not consider > and "" (relational tags).

An error log is generated after completion of the import process. The default location of the log file is:

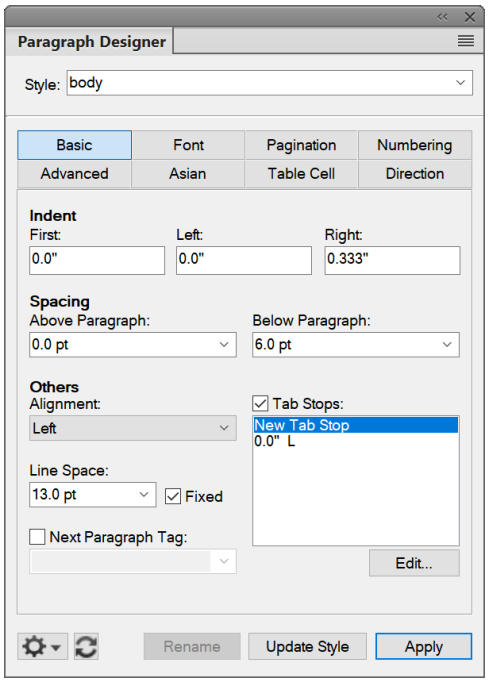
```
%appdata%\adobe\framemaker\fmlogs\17\
```

The logs provide information about the properties that were successfully imported, failed, or ignored by the import process.

> The EDD does not support all properties and selectors defined in CSS 3.0.

Basic Properties

The following screenshot is of the Basic tab properties in the Paragraph Designer:



The following table lists the Basic tab properties mapped to CSS3 properties along with their supported values that can be imported into EDD:

FrameMaker Property	CSS Property	Supported CSS Property Values	Comments/Values
Indent : First	text-indent		The value of this property is equal to the font size.
Indent: Left	margin-left		
Indent: Right	margin-right		
Spacing: Above Paragraph	margin-top		
Spacing: Below Paragraph	margin-bottom		
Alignment	text-align		

FrameMaker Property	CSS Property	Supported CSS Property Values	Comments/Values
Line Space	--fm-line-space	A string value.	These properties are specific to FrameMaker and not a regular CSS3 property.
Fixed Line Space	--fm-line-space	Yes | No	

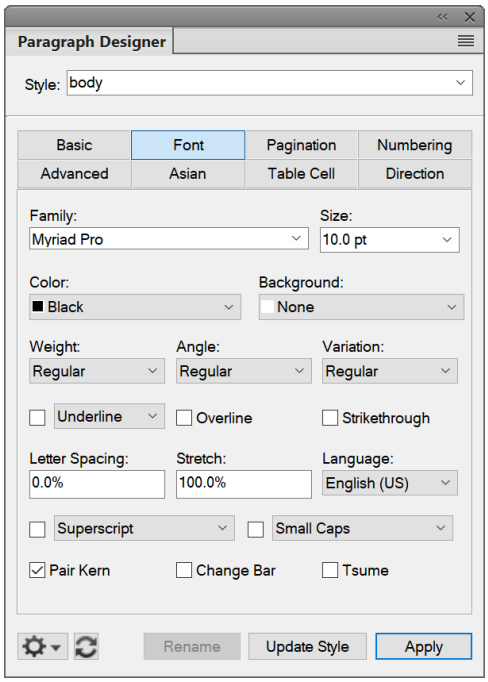
Example:

In the following CSS, the `<shortdesc>` element has been redefined with properties available in the Basic tab:

```
shortdesc{
text-align: justify;
text-indent: 0.8in;
margin-top:2pt;
margin-left:1.2in;
margin-right:2.5in ;
margin-bottom:1pt ;
--fm-line-space:40.5pt;
--fm-allow-line-space:YES;
}
```

Font Properties

The following screenshot is of the Font tab properties in the Paragraph Designer:



The following table lists the Font tab properties mapped to CSS3 properties along with their supported values that can be imported into EDD:

FrameMaker Property	CSS Property	Supported CSS Property Values	Comments/Values
Family	font-family	Only a single font family name, for example: Adobe Clean	

FrameMaker Property	CSS Property	Supported CSS Property Values	Comments/Values
Size	font-size	The following units are supported: - pt (points) - pc (picas) - cm (centimeters) - mm (millimeters) - in (inches) Absolute size with value of: - small - medium - large - x-large - xx-large	The corresponding absolute values in FrameMaker are mapped as follows: - small = 10pt - medium = 12pt - large = 14.4pt - x-large = 17.3pt - xx-large = 20.8pt
Color	color	name|rgb|hsl	- If the name of color is specified and that color is not defined in FrameMaker (CSS includes 16 predefined color names), a new color is created with that name and is assigned the value, rgb. - If the rgb value of the color is specified, a new color name is created with that value.
Background Color	background-color	name|rgb|hsl	
Weight	font-weight	normal | bold	
Angle	font-style	normal | italic |oblique	
Variation	--fm-variation	Condensed | Expanded	This property is specific to FrameMaker and not a regular CSS3 property.
Underline	text-decoration-line:underline	Underline	Numeric Underline is not supported.
Double Underline	text-decoration-style	double underline	
Overline	text-decoration-line:overline	overline	

FrameMaker Property	CSS Property	Supported CSS Property Values	Comments/Values
Strikethrough	text-decoration-line	Strikethrough	
Letter Spacing	--fm-letter-spacing	A numeric value specifying the letter spacing in percentage.	This property is specific to FrameMaker and not a regular CSS3 property.
Stretch	font-stretch	normal | ultra-condensed| extra-condensed| condensed| semi-condensed| semi-expanded| expanded| extra-expanded| ultra-expanded	The corresponding absolute values in FrameMaker are mapped as follows: - Normal=100 - Ultra-condensed=50 - extra-condensed=60 - condensed=72 - semi-condensed=86 - semi-expanded=120 - expanded=144 - extra-expanded=173 - ultra-expanded=207
Superscript/ Subscript	verticle-align	super | sub	
super | sub	font-variant		
Uppercase	text-transform	uppercase	
Lowercase	text-transform	lowercase	
Pair Kern	font-kerning	none | normal	Values are interpreted as: - None=No - Normal=Yes

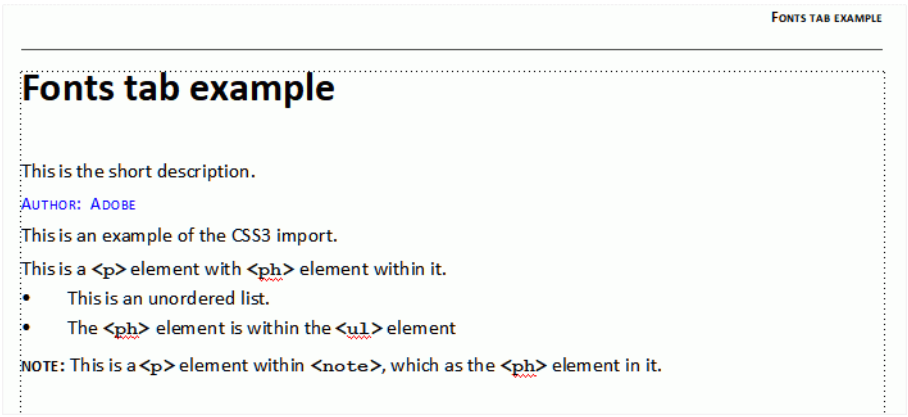
Example:

In the following CSS, the `<ph>` element has been redefined with properties available in the Fonts tab:

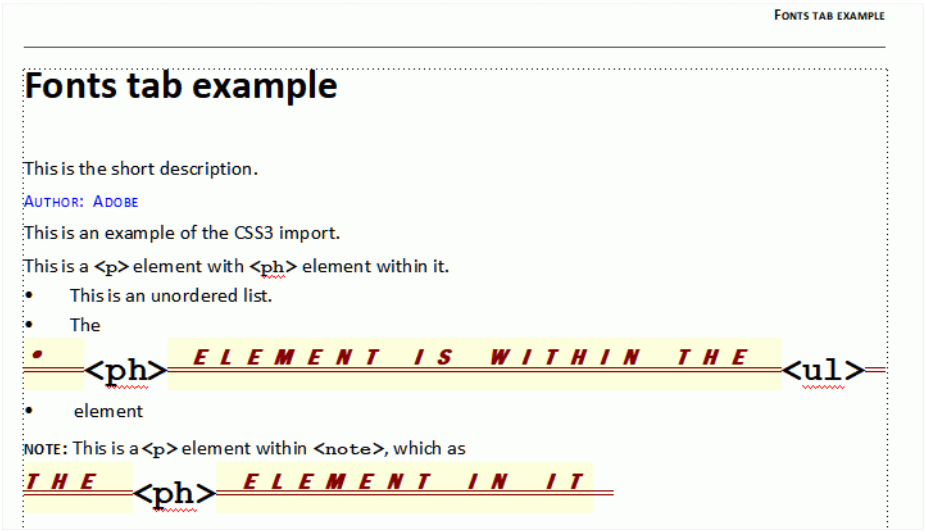
```
ph{
font-family: "Franklin Gothic Demi Cond", "Arial Black", serif;
font-size:xx-large;
color:Maroon;
background-color:rgb(254, 254, 216);
font-style:oblique;
```

```
font-stretch:280;
font-variant:small-caps;
font-weight:bold;
vertical-align: super;
font-kerning: normal;
--fm-letter-spacing:10.32126849878325;
--fm-font-variation:Wide;
text-decoration-line:line-through,overline,underline;
text-decoration-style:double;
}
```

The following screenshot is of a sample file before importing the CSS:

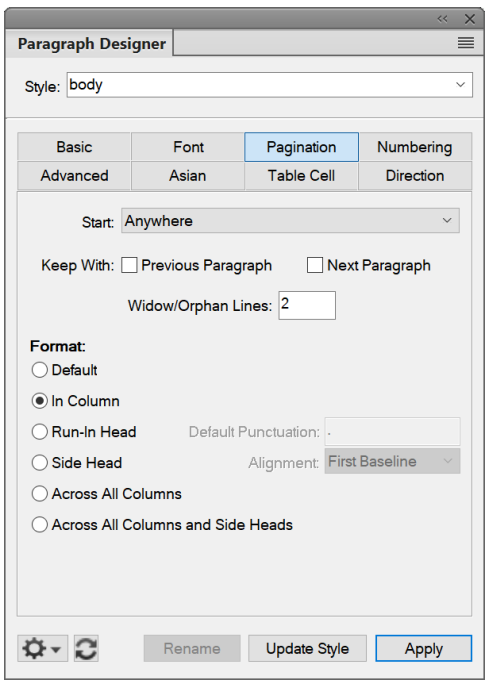


The following screenshot shows the <ph> styling after importing the CSS file:



Pagination Properties

The following screenshot is of the Pagination tab properties in the Paragraph Designer:



The following table lists the Pagination tab properties mapped to CSS3 properties along with their supported values that can be imported into EDD:

FrameMaker Property	CSS Property	Supported CSS Property Values	Comments/Values
Start	page-break-before	auto | always | left | right	Values are interpreted as: - auto=Anywhere - always=Top of page - left=Top of left page - right=Top of right page
Keep With: Previous Paragraph	--fm-allow-previous-paragraph	yes | no	These properties are specific to FrameMaker and not a regular CSS3 property.
Keep With: Next Paragraph	--fm-allow-next-paragraph	yes | no	
Widow/ Orphan Lines	widows/ orphans		
Format: In Column	--fm-allow-in-colum		This property is specific to FrameMaker and not a regular CSS3 property.
Format: Run-In Head	display:run-in		
Format: Side Head	display:compact		
Format: Across All Columns	--fm-allow-across-all-colum		These properties are specific to FrameMaker and not the regular CSS3 properties.
Format: Across All Columns and Side Heads	--fm-allow-across-all-colum-sidehead		

Example:

In the following CSS, the <ph> element has been redefined with properties available in the Pagination tab:

```
ph{
page-break-before:always;
--fm-allow-previous-paragraph:No;
```

```
--fm-allow-next-paragraph:Yes;
orphans:10;
display:run-in;
--fm-run-in-head-default-punctuation:":";
}
```

Advanced Properties

The following table lists the Advanced properties mapped to CSS3 properties along with their supported values that can be imported into EDD:

FrameMaker Property	CSS Property	Supported CSS Property Values	Comments/Values
Automatic Hyphenation	--fm-allow-hyphenate		These properties are specific to FrameMaker and not the regular CSS3 properties.
Maximum Adjacent Hyphens	--fm-max-adjacent-hyphen		
Shortest Word Length	--fm-shortest-word-length		
Shortest Prefix	--fm-shortest-prefix		
Shortest Suffix	--fm-shortest-suffix		
Word Spacing: Allow Automatic Letter Spacing	--fm-allow-letter-spacing		
Word Spacing: Minimum	word-spacing		
Word Spacing: Maximum	--fm-word-spacing-maximum		These properties are specific to FrameMaker and not the regular CSS3 properties.
Word Spacing: optimum	--fm-word-spacing-optimum		
Frame Above Paragraph	--fm-frame-above		
Frame Below Paragraph	--fm-frame-below		
Paragraph Box	--fm-pgf-box-color		

Example:

In the following CSS, the <ph> element has been redefined with properties available in the Advanced tab:

```
ph{
--fm-allow-hyphenate:Yes;
--fm-max-adjacent-hyphen:3;
--fm-shortest-word-length:6;
--fm-shortest-prefix:4;
--fm-shortest-suffix:5;
--fm-allow-letter-spacing:Yes;
--fm-word-spacing-optimum:105;
--fm-word-spacing-maximum:115;
/*--fm-word-spacing-minimum:95;*/
word-spacing:95;
--fm-frame-above:hazard.notice;
--fm-frame-below:hazard.notice;
--fm-pgf-box-color:Yellow;
}
```

Asian Properties

The following table lists the Asian tab properties mapped to CSS3 properties along with their supported values that can be imported into EDD:

FrameMaker Property	CSS Property	Supported CSS Property Values	Comments/Values
Western/ Asian Spacing: Minimum	--fm-western-asian-spacing-minimum		
Western/ Asian Spacing: Maximum	--fm-western-asian-spacing-maximum		
Western/ Asian Spacing: Optimum	--fm-western-asian-spacing-optimum		
Asian Character Spacing: Minimum	--fm-asian-char-spacing-minimum		These properties are specific to FrameMaker and not the regular CSS3 properties.
Asian Character Spacing: Maximum	--fm-asian-char-spacing-maximum		
Asian Character Spacing: Optimum	--fm-asian-char-spacing-optimum		
Squeeze Punctuation	--fm-squeeze-punctuation		
Use Asian Composer	--fm-allow-asian-composer		

Example:

In the following CSS, the <ph> element has been redefined with properties available in the Basic tab:

```
ph{
--fm-western-asian-spacing-minimum:10;
--fm-western-asian-spacing-maximum:60;
--fm-western-asian-spacing-optimum:30;
--fm-asian-char-spacing-minimum:5;
--fm-asian-char-spacing-maximum:15;
```

```
--fm-asian-char-spacing-optimum:6;
/*--fm-squeeze-punctuation:AlwaysSqeeze;*/
/*--fm-squeeze-punctuation:"Always Sqeeze";*/
/*--fm-squeeze-punctuation:"AlwaysSqeeze";*/
--fm-allow-asian-composer:YES;
}
```

CSS Selectors

The CSS selectors are mapped as in the following table:

CSS Selector	Matches
E:first-child	Any element that is the first child of its parent.
:last-child	Any element that is the last child of its parent.
* + E	Any element that not the first element in the document tree.
E + sibling	Any E element that immediately follows its sibling.
E#ID	Attribute context for element with unique ID.

Example:

In the following CSS, the <ph> element has been redefined with the supported CSS selectors:

```
/* descendant selector*/
p ph{
font-family: "Times New Roman";
font-size:x-large;
color:Green;
background-color:rgb(254, 254, 216);
font-style:oblique;
font-stretch:280;
font-kerning: normal;
text-decoration-style:double;
}
/*child selector*/
p>ph{
```

```
font-family:"Arial";
font-size:xx-large;
color:Maroon;
background-color:rgb(254, 254, 216);
font-style:oblique;
font-kerning: normal;
text-decoration-style:Single;
}
/* Immediate sibling*/
p+ph{
font-family: "Times New Roman";
font-size:large;
color:Blue;
background-color:rgb(254, 254, 216);
font-style:oblique;
font-kerning: normal;
text-decoration-line:overline;
}
/*Any Sibling */
p~ph{
font-family: "Times New Roman";
font-size:large;
color:Red;
background-color:rgb(254, 254, 216);
font-style:oblique;
font-kerning: normal;
text-decoration-line:overline;
}
```

Other examples

Some other examples of mapping CSS to EDD are given below.

Prefix and suffix

```
p>note::before{
content: "STARTNOTE";
background-color:Black;
color:Red;}
p>note::after{
content: "ENDNOTE";
background-color:Black;
color:Red;
}
note::after{
content: "ENDNOTE";
background-color:rgb(125, 67, 102);
color:Green;
}
note::before{
content: "STARTNOTE";
background-color:rgb(125, 67, 102);
color:Green;
}
note[outputclass="test"]::before{
content: "TestStart";
background-color:rgb(150, 114, 14);
color:yellow;
}
note[outputclass="test"]::after{
content: "TestEnd";
background-color:rgb(150, 114, 14);
color:yellow;
}
```

Context rules

```
ph[outputclass="left"]{
text-align:left;
color:Red;
font-family:"Helvetica";
font-size:medium;
}
ph[outputclass="center"]{
text-align:left;
color:Red;
background-color:Black;
font-family:"Courier";
font-size:large;
}
ph[outputclass="right"]{
text-align:right;
color:Green;
font-family:"Arial";
font-size:x-large;
}
p >ph:first-child{
background-color: rgb(242, 238, 172);
}
ph:first-child{
background-color:rgb(235, 211, 231);
}
*+ph{
background-color:rgb(150, 114, 14);
}
ph:last-child {
```

```
  background-color:rgb(237, 195, 192);
}
p>ph:last-child {
  background-color:rgb(209, 237, 212);
}
```

Basic level rules

```
li:--fm-level-rule-level(2):--fm-level-rule-start("ul ol"):--fm-level-rule-stop("entry"){
color:Yellow;
background-color:rgb(206, 207, 237);
}
li:--fm-level-rule-level(1):--fm-level-rule-start("ul ol"):--fm-level-rule-stop("entry"){
color:Red;
background-color:rgb(206, 207, 237);
}
li:--fm-level-rule-level(3):--fm-level-rule-start("ul ol"):--fm-level-rule-stop("entry"){
color:Green;
background-color:rgb(206, 207, 237);
}
/* Level Rules with context*/
ul li:--fm-level-rule-level(2):--fm-level-rule-start("ul ol"):--fm-level-rule-stop("entry"){
color:Yellow;
background-color:rgb(244, 245, 225);
}
ul li:--fm-level-rule-level(1):--fm-level-rule-start("ul ol"):--fm-level-rule-stop("entry"){
color:Red;
background-color:rgb(244, 245, 225);
}
ul li:--fm-level-rule-level(3):--fm-level-rule-start("ul ol"):--fm-level-rule-stop("entry"){
color:Green;
background-color:rgb(244, 245, 225);
```

```
}
```

CSS 2 to EDD Mapping

This section provides a reference for the CSS 2 to EDD mapping feature, grouped by CSS property category. Each property's description includes the following headings.

CSS property The CSS 2.0 property name

CSS Property Values A simple list of the available property values.

Mapped to EDD property Shows the element name of the equivalent EDD formatting property. For table parts it shows the mapping for EDD table parts.

Comments/Values Additional information about the mapping includes EDD element property values.

While importing a Cascading Style Sheet (CSS) into an EDD, any property or selector in the CSS that cannot be mapped to an equivalent EDD rule is ignored by FrameMaker. No error log is displayed and errors in the CSS file are not reported.

The EDD does not support all properties and selectors defined in CSS 2.0. While importing a CSS into an EDD, FrameMaker will ignore any unsuported properties or selectors.

CSS Font Properties

Fonts are mapped as in the following table:

CSS property	CSS Property Values	Mapped to EDD property	Comments/Values
`font-family`	`font-family` `family-name | generic-family`	`PropertiesFont < Family` element.	`Font Set` is not supported. • `Generic-Family` can't be supported. • Only one `font-family` can be specified using the EDD `Family` element.
`font-size`	length in units • `cm` (centimeters) • `ex` (exs) • `in` (inches) • `mm` (millimeters) • `pc` (picas) • `pt` (points) • `px` (pixels)	`PropertiesFont < Size` element.	Only font-size with a length in points is recognised, all other length types are ignored and the

CSS property	CSS Property Values	Mapped to EDD property	Comments/Values
	% (percentage)		% values are not mapped as FrameMaker does not calculate relative values proportionally.
	Relative size with these values: • larger • smaller	Not supported	
	Absolute size with value of: • xx-small • x-small • small • medium • large • x-large • xx-large	PropertiesFont < Size element.	The corresponding absolute values in FrameMaker are mapped as follows: • xx-small = 7.0pt • x-small = 8.4pt • small = 10pt • medium = 12pt (Default) • large = 14.4pt • x-large = 17.3pt • xx-large = 20.8pt
font-style	normal|italic|oblique	PropertiesFont < Angle with Regular or Italic child elements. CSS oblique is mapped to EDD Italic.	
font-variant	normal|small-caps	CSS small-caps is mapped to EDD PropertiesFont < Case < SmallCaps. No action for normal.	
font-weight	normal|bold|bolder|lighter|100|200|300|400|500|600|700|800|900	PropertiesFont < Weight. CSS normal and bold are mapped to Regular and Bold All weights <= 400 are mapped to Regular, and > 400 are mapped to Bold.	Relative values bolder and lighter cannot be mapped as FrameMaker does not calculate relative values.

CSS property	CSS Property Values	Mapped to EDD property	Comments/Values
font	font-style|font-variant|font-weight|font-size line-height|font-family|caption|icon|menu|message-box|small-caption|status-bar	As listed for the individual CSS properties above.	caption, icon, menu, message-box, small-caption and status-bar fonts are not supported.
font-stretch	normal|ultra-condensed|condensed|semi-condensed|semi-expanded|expanded|extra-expanded|ultra-expanded	PropertiesFont < Stretch.	The mappings from CSS to FrameMaker EDD are: • ultra-condensed = 50 • extra-condensed = 60 • condensed = 72 • semi-condensed = 86 • normal = 100 • semi-expanded = 120 • expanded = 144 • extra-expanded = 173 • ultra-expanded = 207
	wider|narrower	PropertiesFont < StretchChange.	• wider = +20 • narrower = -20
font-size-adjust	number|none		@font-face is not supported.

CSS text properties

The CSS text properties are mapped as in the following table:

CSS property	CSS Property Values	Mapped to EDD property	Comments/Values
text-decoration	underline	PropertiesFont < Underline element.	
	overline	PropertiesFont < Overline element.	
	line-through	PropertiesFont < Strikethrough element.	

CSS property	CSS Property Values	Mapped to EDD property	Comments/Values
	blink	Blink is not supported.	
text-transform	uppercase	PropertiesFont < Case < Uppercase element.	Both text-transform and font-variant map to the Case element of EDD. If both these properties are used for an element context, then only the text-transform value is used.
	lowercase	PropertiesFont < Case < Lowercase element.	
	capitalize	capitalize is not supported.	
text-align	left|right|center|justify|string	PropertiesBasic < PgfAlignment. CSS left, right, center and justify are mapped to EDD Left, Right, Center, and Justified respectively. CSS string is not supported.	
text-indent	length|percentage	PropertiesBasic < Indents < FirstIndent percentage value is not supported.	
line-height	number|length|percentage	Not supported	
word-spacing	normal|length|inherit	PropertiesAdvanced < WordSpacing. The CSS length value maps to the EDD minimum value.	In the EDD, WordSpacing accepts percentage values of the font's em space. Therefore, only an em value of CSS word-spacing can be mapped to EDD.
letter-spacing	normal|length|inherit	PropertiesAdvanced < LetterSpacing.	In the EDD, LetterSpacing can have a value of "yes" or "no". A positive value for CSS length maps to "yes" in the EDD.
text-shadow		Not supported	
white-space		Not supported	

CSS color and backgrounds properties

The CSS color and background properties are mapped as in the following table:

CSS property	CSS Property Values	Mapped to EDD property	Comments/Values
color	name|rgb	PropertiesFont < Color	• If the name of color is specified and that color is not defined in FrameMaker (CSS includes 16 predefined color names), a new color is created with that name and is assigned the value, rgb. • If the rgb value of the color is specified, a new color name is created with that value.
background-color		background color	
background-image		Not supported	
background-attachment		Not supported	
background-position		Not Supported	
background-repeat		Not Supported	
background	background-color*|background-image|background-repeat|background-attachment|background-position *Mapped to EDD property, background color.		

CSS Formatting Model

The CSS Box Model and Formatting Model are mapped as in the following table:

CSS property	CSS Property Values	Mapped to EDD property	Comments/Values
margin-right	length|percentage|auto	PropertiesBasic < Indents < RightIndent. The percentage value is not supported.	
margin-left	length|percentage|auto	PropertiesBasic < Indents < FirstIndent and LeftIndent. The percentage value is not supported.	
margin-top	length|percentage|auto	PropertiesBasic < Indents < SpaceAbove. The percentage value is not supported.	
margin-bottom	length|percentage|auto	PropertiesBasic < Indents < SpaceBelow. The percentage value is not supported.	
margin	margin-right|margin-left|margin-top|margin-bottom	As listed for the individual CSS properties above.	
border, border*		Not supported	
padding, padding*		Not supported	
width		Not supported	
height		Not supported	
min-width		Not supported	
min-height		Not supported	
max-width		Not supported	
max-height		Not supported	

CSS property	CSS Property Values	Mapped to EDD property	Comments/Values
`float`	`left`	`PropertiesPagination < Placement < SideHead < Left`	The main flow in the target structured document must have “room for side head” enabled to acheive the expected result.
	`right`	Not supported	
`clear`		Not supported	

CSS property	CSS Property Values	Mapped to EDD property	Comments/Values
display	inline	CSS inline elements are supported by using the TextRangeFormatting element.	• When CSS inline is specified all CSS properties that map to EDD PropertiesFont child elements are retained. All other CSS properties are ignored. • The default behaviour of CSS inline is equivalent to a FrameMaker text range. • The default behaviour of CSS block is equivalent to a FrameMaker paragraph. • If there are two different rules for a single element in which one of the selectors is more specific than the other, and both rules specify the display property with a different value, then in FrameMaker the final value of the display property is undefined, and the corresponding element type in the EDD is also undefined.
	block	CSS block elements are supported by using the ParagraphFormatting element.	
	run-in	PropertiesPagination < Placement < RunInHead element	
	compact	PropertiesPagination < Placement < SideHead element	The main flow in the target structured document must have “room for side head” enabled to acheive the expected result.
	list-item	PropertiesNumbering < AutoNumFormat	An appropriate AutoNumFormat must be created based on the list-style-type property.

CSS Pagination Properties

The CSS Pagination properties are mapped as in the following table:

CSS property	CSS Property Values	EDD property	Comments/Values
page-break-before	auto|always|avoid|left|right|inherit	PropertiesPagination < StartPosition CSS to EDD element mapping: • always = TopOfpage • left = TopOfLeftPage • right = TopOfRightPage. The avoid property is not supported.	
page-break-after		Not supported	
page-break-inside		Not supported	
widows/ orphans	integer|inherit	PropertiesPagination < WidowOrphanLines	In CSS, widows and orphans are different properties and hence they can have different values. But, in the EDD, a single element, WidowOrphanLines, controls both values, and hence they have the same value.
marks	crop|cross	Not supported	
@page		Not supported	An EDD has no control over the page layout. In FrameMaker page layout is designed into the structured template.
page		Not supported	To acheive the required result set up a suitable **ApplyMasterPages** command. See the Using Adobe® FrameMaker® guide.
size	length|auto|portrait|landscape|inherit		An EDD has no control over the page layout. In FrameMaker page layout is designed into the structured template.

CSS generated content, automatic numbering, and lists

The CSS generated content, automatic numbering, and lists are mapped as in the following table:

CSS property	CSS Property Values	Mapped to EDD property	Comments/Values
`list-style-type`	• disc • circle • square • decimal • decimal-leading-zero • lower-roman • upper-roman • lower-alpha • upper-alpha • lower-latin • upper-latin • lower-greek • hebrew • armenian • georgian • cjk-ideograph • hiragana • katakana • hiragana-iroha • katakana-iroha • none	Not supported	This CSS property is not supported in EDD. We have to enhance EDD for this.
`list-style-image`		Not supported	
`list-style-position`		Not supported	
`list-style`	`list-style-type`, `list-style-image`, `list-style-position`	Not supported	

CSS property	CSS Property Values	Mapped to EDD property	Comments/Values
content	string	The text content of the Prefix or Suffix element.	string, attr(attname), open-quote and close-quote may be used in any combination as required. Separate each item with whitespace. • To create a Prefix use the CSS :before psuedo element selector • To create a Suffix use the CSS :after psuedo element selector • In CSS, the string generated by the content property can have any CSS style. In contrast, EDD Prefix and Suffix rules can have only use font formatting (through the PropertiesFont element).
	attr(attname)	<$attribute[attname]>	
	open-quote	"	
	close-quote	"	
	counter	Not supported	
	uri, quotes	Not supported	
counter-increment		Not supported	
counter-reset		Not supported	
counter		Not supported	
counters		Not supported	
marker		Not supported	
marker offset		Not supported	
White-space		Not supported	
position		Not supported	
z-index		Not supported	
visibility		Not supported	

CSS Tables

`Container` is the default element type in an EDD. An element can be specified in CSS as a `table component` or `table component group` using the `display` property. If an element is a `Container` in the EDD but the CSS specifies the element as Table/table-Tow, then the element type in EDD is changed from `Container` to the corresponding table element type.

CSS property	CSS Property Values	Mapped to EDD property	Comments/Values
`display`	`table`	`Element < Table`	• FrameMaker table part elements cannot have `PrefixRules` or `SuffixRules`. So, a rule with the `:after` or `:before` pseudo element selector, is ignored. • FrameMaker table part elements cannot have `TextRangeFormatting` element in the EDD. So, the `inline` value of the `display` property is ignored.
	`table-inline`	Not supported	
	`table-row`	`Element < TableRow`	
	`table-row-group`	`Element < TableBody`	
	`table-header-group`	`Element < TableHeading`	
	`table-footer-group`	`Element < TableFooting`	
	`table-cell`	`Element < TableCell`	
	`table-caption`	`Element < TableTittle`	
	colspan rowspan	Straddling in FrameMaker core	The New element needs to be added in EDD.
	border	FM core supports border in Table and Table cell.	The New element needs to be added in EDD.
	background	FM core supports background in Table and Table cell.	The New element needs to be added in EDD.

CSS property	CSS Property Values	Mapped to EDD property	Comments/Values
	table-column table-column-group	Not supported. Column selectors are also not supported as they are applied in table-column and table-column-group only.	
caption-side	top|bottom|left|right	Not supported	
empty-cells	show|hide	Not supported	
table-layout	auto|fixed	Not supported	

CSS Selectors

The CSS selectors are mapped as in the following table:

CSS selector	Matches	EDD selector
*	any element	The * selector matches any single element of the document tree. So, properties specified using * are applied to all elements in EDD.
E	Any element Elem	**Element(Container):** E
F E	Any element E that is descendent of element F	**If context is:** * < F
F > E	Any E element that is child of F	**Element (Container):** E **If context is:** F
F + E	Any E element that immediately follows F	{after F} * + E maps to {notfirst}.
.class	any element with class "class"	Not supported
#id	element with ID id	**Element (Container):** E **If context is:** [IDname="id"]
:first-child	Any element that is the first child of its parent	{first}
:link :visited	Hyperlink visited or not	Ignored as it does not apply to FrameMaker.

CSS selector	Matches	EDD selector
:active :hover :focus	Any element that is activated by the user using the mouse, etc.	Ignored as it is for an interactive browser
:lang(c)	Any element whose content is in the 'C' language	**Element (Container):**E **General Rule:** <ANY> **If context is:** [xml:lang="c"]
[att]	Any element with attribute att	Not supported
[att=val]	Any element with attribute att and value val.	**Element (Container):**E **General Rule:** <ANY> **If context is:** [att="val"]
[att~=val]	Any element that includes the word "val" in its value.	Not supported
[att |= "val"]	Any element with an att attribute value "val-.."	Not supported
E:first-letter	The first letter of any element E	Not supported
E:first-line	The first line of any element E	Not supported
E:before E:after	The text to be inserted at the start/end of any element E	Maps to Prefix and Suffix rules in EDD. For more details, see the "content" property.

6 XML Schema to DTD Mapping

When XML documents are associated with an XML Schema declaration, FrameMaker can convert the Schema to a DTD declaration, from which you can create or modify an EDD. The content models of Schema and DTD are not identical. This chapter shows how Schema definitions are mapped into DTD definitions.

For details of how special objects are handled when converting Schema to DTD, see the individual object discussions in Developer Guide, Part IV, Translating Between Markup Data and FrameMaker

> **Note:** The DTD generated from Schema always uses UTF8 encoding, regardless of the encoding used in the Schema file.

If you wish to modify the DTD that is generated automatically, you can do so. If you do this, reference the modified DTD from the original XML document. When FrameMaker imports an XML document that references both a Schema and DTD, it uses the DTD to create the FrameMaker elements, although it still validates the contents against the Schema.

Schema location

You can import an XML document that references a Schema file, and you can specify a Schema file in your structure application, to use for validating a document upon export to XML.

To specify a Schema file for use in exporting to XML, modify the `structapps.fm` file. The element `Schema`, a child of the `XmlApplication` element, specifies the Schema file path for export. The property `Namespace` in `XmlApplication` must be set to true if instance documents use namespaces. See "Specifying a Schema for XML" on page 23

For importing an XML document, include the path of the Schema file in the XML using attributes—`noNamespaceSchemaLocation` or `schemaLocation` depending on whether your schema includes a target namespace or not. A DTD is generated automatically when you import the XML, and the EDD is generated from the DTD.

Schema allows an XML document to reference multiple Schema locations in different namespaces using the root-element attribute `xsi:schemaLocation`, which can have multiple values. This feature has no equivalent in DTD. If an XML document references multiple Schema locations, FrameMaker uses only the first one for generating a DTD and for validation.

You can load XML documents that use `noNamespaceSchemaLocation`. For example:

```
<RootElementName id="RootElementID"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MySchema.xsd">
```

If an imported document references both a valid DTD location and a Schema location, the document is validated against the Schema. If there is no Schema location value, it is validated against the DTD. If neither location is specified, the load shows a warning similar to the one for a document that has no `DOCTYPE` statement.

If an imported document references both a DTD location and a Schema location, but the referenced DTD location is not valid, the load fails with the error “invalid external entity.” FrameMaker does not, in this case, generate a new DTD from the referenced Schema.

Namespace and Schema location attributes

The root element is not created automatically, therefore, the conversion process adds attributes for namespace definitions and schema location in all global elements specified in the Schema, which are then copied into the EDD that is created from the DTD. If you do add a root element, as recommended, these attributes are not needed, although they are not harmful.

If you wish, you can remove these extra attributes in two ways:

- After you generate an EDD from Schema, remove the extra attributes from the non-root elements in the EDD, and create a template. In this case, you do not need to provide an external DTD in the instance XML document.
- Remove the extra attributes from the non-root elements in the generated DTD, and save the modified DTD as an external DTD in the instance document. This is the technique to use if you want to modify the default mapping to DTD. In this case, you do not need a template. If you do wish to create a template, you can remove the attributes from the EDD as well.

Simple type mapping

All simple types in Schema translate to `#PCDATA` in DTD, and the Schema type `anyType` translates to the DTD type `ANY`. For example:

Schema

```
<xsd:element name="AString" type="xsd:string"/>
<xsd:element name="AnUnsignedInt" type="xsd:unsignedInt"/>
<xsd:element name="ABoolean" type="xsd:boolean"/>
<xsd:element name="AgYearMonth" type="xsd:gYearMonth"/>
<xsd:element name="AgMonthDay" type="xsd:gMonthDay"/>
<xsd:element name="AnyTypeElem" type="xsd:anyType"/>
```

DTD

```
<!ELEMENT AString(#PCDATA)>
<!ELEMENT AnUnsignedInt(#PCDATA)>
<!ELEMENT ABoolean   (#PCDATA)>
<!ELEMENT AgYearMonth (#PCDATA)>
<!ELEMENT AgMonthDay (#PCDATA)>
<!ELEMENT AnyTypeElem (ANY)>
```

Attributes of simple type elements

Attribute of simple types translate to `CDATA`, `NMTOKEN`, `NMTOKENS`, `ID`, `IDREFS`, `ENTITY`, and so on. Enumeration facets in attributes are exported to DTD. Other simple type facets, `xsd:list` facets, and `xsd:union` facets are dropped.

Note the translation of `use`, `fixed` and `default` attribute combinations in the following example:

Schema

```
<xsd:attribute name="ReqdAttr" type="xsd:int" use="required"/>
<xsd:attribute name="OptAttr" type="xsd:int" use="optional"/>
<xsd:attribute name="ProhAttr" type="xsd:int" use="prohibited"/>
<xsd:attribute name="FixedReqdAttr" type="xsd:int" use="required"
fixed="23"/>
<xsd:attribute name="OptDefAttr" type="xsd:int" use="optional"
default="12"/>
<xsd:attribute name="FixedOptAttr" type="xsd:int" use="optional"
fixed="25"/>
<xsd:attribute name="EnumAttr" use="optional" default="Male">
<xsd:simpleType><xsd:restriction base="xsd:string">
   <xsd:enumeration value="Male"/>
   <xsd:enumeration value="Female"/>
</xsd:restriction></xsd:simpleType>
</xsd:attribute>
```

DTD

```
 <!ATTLIST ElemName
      FixedOptAttr  NMTOKEN   #FIXED "25"
      EnumAttr  (Male|Female)  "Male"
      OptDefAttr  NMTOKEN   "12"
      ReqdAttr  NMTOKEN   #REQUIRED
      FixedReqdAttr  NMTOKEN   #FIXED "23"
      OptAttr   NMTOKEN   #IMPLIED>
```

Complex type mapping

Complex content models in Schema translate to similar constructs in DTD, insofar as possible. If there are any errors in the Schema that result in a content model ambiguity, the content model is translated to `ANY` in DTD.

Group

The `group` content model in Schema translates to a group in DTD. For example:

Schema

```
<xsd:element name="GroupElem">
<xsd:complexType><xsd:sequence><xsd:choice>
   <xsd:group ref="IntStr"/>
   <xsd:element name="MMIncl" type="xsd:string"/>
</xsd:choice></xsd:sequence></xsd:complexType>
</xsd:element>

<xsd:group name="IntStr" id="Group1">
<xsd:sequence>
   <xsd:element name="Int" type="xsd:int" minOccurs="2"
             maxOccurs="2"/>
   <xsd:element name="Str" type="xsd:string"/>
</xsd:sequence>
</xsd:group>
```

DTD

```
<!ELEMENT GroupElem (((abc:Int,abc:Int),abc:Str)|abc:MMIncl)>
```

Sequence

A Schema `sequence` content model translates to a sequence in DTD. Note the translation of `minOccurs` and `maxOccurs` attribute value combinations in the following example.

Schema

```
<xsd:element name="TestOccurence">
<xsd:complexType>
<xsd:sequence>
   <xsd:element name="Min0" type="xsd:int" minOccurs="0"/>
   <xsd:element name="Max1" type="xsd:int" maxOccurs="1"/>
   <xsd:element name="Min0Max1" type="xsd:int" minOccurs="0"
      maxOccurs="1"/>
   <xsd:element name="Min1Max1" type="xsd:int" minOccurs="1"
       maxOccurs="1"/>
   <xsd:element name="Min2MaxI" type="xsd:int" minOccurs="2"
      maxOccurs="unbounded"/>
   <xsd:element name="Min0Max2" type="xsd:int" minOccurs="0"
      maxOccurs="2"/>
   <xsd:element name="Min2Max10" type="xsd:int" minOccurs="2"
      maxOccurs="10"/>
   <xsd:element name="Min2Max3" type="xsd:int" minOccurs="2"
      maxOccurs="3"/>
</xsd:sequence>
</xsd:complexType>
```

DTD

```
<!ELEMENT TestOccurence
((Min0?,Max1,Min0Max1?,Min1Max1,(Min2MaxI,Min2MaxI,Min2MaxI*),
 (Min0Max2*),(Min2Max10,Min2Max10,Min2Max10*),
 (Min2Max3, Min2Max3, Min2Max3?))>
```

Choice

A Schema `choice` content model translates to a choice in DTD. For example:

Schema

```
<xsd:element name="ChoiceElem">
<xsd:complexType>
   <xsd:sequence>
      <xsd:choice>
         <xsd:element name="Int" type="xsd:int"/>
         <xsd:element name="Str" type="xsd:string"/>
         <xsd:element name="MMIncl" type="xsd:int"/>
      </xsd:choice>
   </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

DTD

```
<!ELEMENT ChoiceElem ((Int|Str)|MMIncl)>
<!ELEMENT Int        (#PCDATA)>
<!ELEMENT Str        (#PCDATA)>
<!ELEMENT MMIncl     (#PCDATA)>
```

All

A Schema `all` content model translates to a choice of elements with multiple occurrences in DTD. For example:

Schema

```
<xsd:element name="DataType">
<xsd:complexType>
<xsd:all>
<xsd:element name="AName" type="xsd:Name"/>
<xsd:element name="AQName" type="xsd:QName"/>
<xsd:element name="ANCName" type="xsd:NCName"/>
<xsd:element name="AnyURI" type="xsd:anyURI"/>
<xsd:element name="ALanguage" type="xsd:language"/>
<xsd:element name="AnID" type="xsd:ID"/>
<xsd:element name="AnIDRef" type="xsd:IDREF"/>
<xsd:element name="AIDREFS" type="xsd:IDREFS"/>
</xsd:all></xsd:complexType></xsd:element>
```

DTD

```
<!ELEMENT DataType
(AName|AQName|ANCName|AnyURI|ALanguage|AnID|AnIDRef|AIDREFS)*>
```

Named complex types

Named complex types in Schema are dropped, and their content model is substituted into the corresponding DTD elements. For example:

Schema

```
<xsd:element name="AddressDetails">
<xsd:complexType><xsd:sequence>
   <xsd:element name="ToAddress" type="USAddress"/>
   <xsd:element name="FromAddress" type="USAddress"/>
</xsd:sequence></xsd:complexType>
</xsd:element>

<xsd:complexType name="USAddress">
<xsd:sequence>
   <xsd:element name="name" type="xsd:string"/>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="city" type="xsd:string"/>
   <xsd:element name="state" type="xsd:string"/>
   <xsd:element name="zip" type="xsd:int"/>
</xsd:sequence>
<xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
<xsd:attribute name="headquarter" type="xsd:string"
               use="required"/>
</xsd:complexType>
```

DTD

```
<!ELEMENT AddressDetails (ToAddress,FromAddress)>
<!ELEMENT ToAddress  ((name,street,city,state),zip)>
<!ATTLIST ToAddress  country   NMTOKEN   #FIXED "US"
                  headquarter  CDATA     #REQUIRED >
<!ELEMENT FromAddress
                  ((name,street,city,state),zip)>
<!ATTLIST FromAddress
                  country   NMTOKEN   #FIXED "US"
                  headquarter  CDATA     #REQUIRED >
<!ELEMENT name       (#PCDATA)>
<!ELEMENT street     (#PCDATA)>
<!ELEMENT city       (#PCDATA)>
<!ELEMENT state      (#PCDATA)>
<!ELEMENT zip        (#PCDATA)>
```

Named attribute groups

Named attribute groups in Schema are dropped, and the attributes are put into the corresponding DTD attribute list. For example:

Schema

```
<xsd:element name="PersonalDetails">
<xsd:complexType>
   <xsd:sequence>
      <xsd:element name="Name" type="xsd:string"/>
   </xsd:sequence>
   <xsd:attributeGroup ref="PersonalData"/>
</xsd:complexType>
</xsd:element>

<xsd:attributeGroup name="PersonalData">
<xsd:attribute name="Age" type="xsd:int" use="required"/>
<xsd:attribute name="Gender">
   <xsd:simpleType><xsd:restriction base="xsd:string">
      <xsd:enumeration value="Male"/>
      <xsd:enumeration value="Female"/>
   </xsd:restriction></xsd:simpleType>
</xsd:attribute>
</xsd:attributeGroup>
```

DTD

```
<!ELEMENT PersonalDetails (Name)>
<!ATTLIST PersonalDetails
                 Age   NMTOKEN   #REQUIRED
                 Gender  (Male|Female)  #IMPLIED>
<!ELEMENT Name (#PCDATA)>
```

Abstract elements

For an abstract `element` in Schema is substituted into DTD elements using its own substitution group, if one is defined. Otherwise, the element maps directly to a DTD element. For example:

Schema

```
<xsd:element name="RootElem">
<xsd:complexType><xsd:all>
   <xsd:element name="Elem1" type="xsd:int" minOccurs="0"/>
   <xsd:element ref="AbstractElem"/>
</xsd:all></xsd:complexType>
</xsd:element>

<xsd:element name="AbstractElem" type="xsd:string"
          abstract="true"/>
<xsd:element name="Substitute1" type="xsd:string"
          substitutionGroup="AbstractElem"/>
<xsd:element name="Substitute2" type="xsd:string"
          substitutionGroup="AbstractElem"/>
```

DTD

```
<!ELEMENT RootElem (Elem1?|(Substitute1|Substitute2))*>
<!ELEMENT Elem1  (#PCDATA)>
<!ELEMENT Substitute1 (#PCDATA)>
<!ELEMENT Substitute2 (#PCDATA)>
```

Mixed content models

A `mixed` content model translates to a multiple occurence of choice between elements in the content model and `#PCDATA`. Occurence constraints associated with the elements and content model are ignored. For example:

Schema

```
<xsd:element name="RootElem">
<xsd:complexType mixed="true"><xsd:sequence>
   <xsd:element name="elem1" type="xsd:string"
       maxOccurs="unbounded"/>
   <xsd:element name="elem2" type="xsd:positiveInteger"/>
   <xsd:element name="elem3" type="xsd:string"/>
   <xsd:element name="elem4" type="xsd:date" minOccurs="0"/>
</xsd:sequence></xsd:complexType>
</xsd:element>
```

DTD

```
<!ELEMENT RootElem   (#PCDATA|elem1|elem2|elem3|elem4)*>
<!ELEMENT elem1      (#PCDATA)>
<!ELEMENT elem2      (#PCDATA)>
<!ELEMENT elem3      (#PCDATA)>
<!ELEMENT elem4      (#PCDATA)>
```

Supported Schema features

Supported element qualification features of Schema are listed below with their mapping into DTD.

Defaults

The Schema `attributeFormDefault` and `elementFormDefault` are honored wherever they occur. For example:

Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="Schema-abstract-elements"
xmlns:abc="Schema-abstract-elements"
elementFormDefault="qualified">

<xsd:element name="RootElem">
<xsd:complexType><xsd:sequence>
   <xsd:element name="Elem1" type="xsd:int"/>
</xsd:sequence></xsd:complexType>
</xsd:element>
```

DTD

```
<!ELEMENT abc:RootElem (abc:Elem1)>
```

Any

Any content model containing the Schema `<any>` element translates to the DTD `ANY` content model, regardless of additional content. For example:

Schema

```
<xsd:element name="AnyElem">
<xsd:complexType><xsd:sequence>
   <xsd:element name="Elem1" type="xsd:int"/>
   <xsd:any namespace="http://www.w3.org/1999/xhtml"
         minOccurs="1" maxOccurs="unbounded"
         processContents="skip"/>
</xsd:sequence></xsd:complexType>
</xsd:element>
```

DTD

```
<!ELEMENT AnyElem ANY>
<!ELEMENT Elem1 (#PCDATA)>
```

Notice in this example that the `Elem1` element is translated independently, and is not part of `AnyElem` in the DTD.

Extension and restriction of complex types

Extension and restriction of a complex type in Schema translates directly to the DTD. For example:

Schema

```
<xsd:element name="ElemA" type="ComplexTypeB"/>
<xsd:complexType name="ComplexTypeA">
<xsd:sequence>
   <xsd:element name="elem1" type="xsd:string" maxOccurs="3"/>
   <xsd:element name="elem2" type="xsd:string"/>
</xsd:sequence>
<xsd:attribute name="attr1" type="xsd:NMTOKEN"/>
<xsd:attribute name="attr2" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="ComplexTypeB">
<xsd:complexContent>
   <xsd:extension base="ComplexTypeA">
      <xsd:attribute name="attr3" type="xsd:date" use="required"/>
   </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

DTD

```
<!ELEMENT ElemA      ((elem1,elem1*),elem2)>
<!ATTLIST ElemA      attr1     NMTOKEN   #IMPLIED
                     attr2     CDATA     #REQUIRED
                     attr3     NMTOKEN   #REQUIRED >
<!ELEMENT elem1      (#PCDATA)>
<!ELEMENT elem2      (#PCDATA)>
```

Include, import, and redefine

The `include`, `import` and `redefine` constructs allow one Schema file to refer to other Schema files. In converting to DTD, information from such referenced Schema files is included, but all elements are output to a single DTD. For example, if a Schema file `a.xsd` with namespace `ns_a` imports another Schema, `b.xsd` with namespace `ns_b`, the resulting DTD contains elements from both `ns_a` and `ns_b` namespaces.

The following example shows three Schema files; the first, `example.xsd`, includes the file named `include.xsd`, and imports the file named `import.xsd`. When the file `example.xsd` is imported into FrameMaker, the resulting DTD includes definitions for all three files.

Schema

First file, example.xsd

```
<schema targetNamespace="Include-Import-Example"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:a="Include-Import-Example"
   xmlns:b="Import-schema" elementFormDefault="qualified">
<include schemaLocation="./include.xsd"/>
<import namespace="Import-schema" schemaLocation="./import.xsd"/>
<element name="rootElem1">
<complexType><sequence>
   <element name="elem1" type="a:complexTypeA"/>
   <element ref="b:importElem1"/>
   <element ref="a:includeElem3"/>
   </sequence>
</complexType>
</element>
</schema>
```

Second file, include.xsd

```
<schema targetNamespace="Include-Import-Example"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:a="Include-Import-Example"
   elementFormDefault="qualified">
<complexType name="complexTypeA"><sequence>
<element name="includeElem1" type="string"/>
<element name="includeElem2" type="string"/>
</sequence></complexType>
<element name="includeElem3" type="int"/>
</schema>
```

Third file, import.xsd

```
<schema targetNamespace="Import-schema"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:c="Import-schema" elementFormDefault="qualified">
<element name="importElem1" type="int"/>
<element name="importElem2" type="string"/>
</schema>
```

DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT c:importElem1 (#PCDATA)>
<!ELEMENT c:importElem2 (#PCDATA)>
<!ELEMENT includeElem1  (#PCDATA)>
<!ELEMENT includeElem2  (#PCDATA)>
<!ELEMENT includeElem3  (#PCDATA)>
<!ELEMENT rootElem1  ((elem1,c:importElem1),includeElem3)>
<!ELEMENT elem1      (includeElem1,includeElem2)>
```

Unsupported Schema features

Features of Schema listed below cannot be mapped into DTD, and are dropped:

- Abstract types
- `key`, `keyref`, and `unique`
- Annotations

7 The CALS/OASIS Table Model

The CALS or the related OASIS table model is a specific set of element and attribute declarations for defining tables, originally defined in "Markup Requirements and Generic Style Specification for Electronic Printed Output and Exchange of Text," MIL-M-28001B. The OASIS table model is an XML expression of the exchange subset of the full CALS Table Model DTD. If your markup documents use these elements and attributes or some simple variations of them, FrameMaker can translate them to tables and table parts without the assistance of read/write rules. The CALS model can be interpreted in various ways. This chapter describes the CALS elements and attributes as they are interpreted by FrameMaker.

Some attributes are common to several elements in the description of the table. In these cases, attribute values are inherited in the element hierarchy. The values of attributes associated with `<colspec>` and `<spanspec>` elements act as though they were on the parent element for inheritance purposes. This is, if a `<tgroup>` element has two `<colspec>` child elements and a `<thead>` child element, the attributes of the `<colspec>` elements apply to the `<thead>` element unless that element has its own `<colspec>` elements with attribute values that override the inherited ones. If you want to change how FrameMaker processes any attribute of a `<colspec>` or `<spanspec>` element, you refer to the attribute as a formatting property.

In the CALS model, the `<table>` element has an `<orient>` attribute. This attribute is not supported in FrameMaker, because there is no way in a FrameMaker table to specify orientation on the page.

FrameMaker properties that DO NOT have corresponding CALS attributes

FrameMaker Property	For FrameMaker Elements of Type	Corresponding CALS Attribute
column widths	table (CALS: tgroup)	(none)

Column widths: Width of successive columns in the table. Each value is either an absolute width or a width proportional to the size of the entire table. If proportional widths are used, the CALS -pgwide- attribute determines the table width. For example, to specify that the first two columns are each one-quarter the size of the table, and the third column is half the size of the table, you could write a rule to specify your column widths as "25* 25* 50*". Valid units and abbreviations for the "column width" formatting property are:

Unit	Abbreviation
centimeter	cm
cicero	cc
didot	dd
inch	in (in FrameMaker dialog boxes, " is also used, but not for "column width" formatting property)
millimeter	mm
pica	pc (or pi)
point	pt

FrameMaker Property	For FrameMaker Elements of Type	Corresponding CALS Attribute
maximum height	row	(none)
Maximum height of a row in a table.		
minimum height	row	(none)
Minimum height of a row in a table.		
row type	row	(none)
Whether the associated table row is a heading, footing, or body row, or the associated table cell occurs in a row of that type.		
horizontal straddle	cell	(none)
How many columns this straddle cell spans		
vertical straddle	cell	(none)
How many rows this straddled cell spans		

Element and attribute definition list declarations

The element and attribute declarations as used by FrameMaker are as follows:

```
<!ENTITY % yesorno "NUMBER">
```

```
<!ELEMENT table - - (title?, tgroup+)>
<!ATTLIST table
 colsep   %yesorno; #IMPLIED
   frame    (all|top|bottom|topbot|sides|none) #IMPLIED
   pgwide   %yesorno; #IMPLIED
   rowsep   %yesorno; #IMPLIED
   tabstyle NMTOKEN   #IMPLIED
>

<!ELEMENT title - - (#PCDATA)>

<!ELEMENT tgroup - O (colspec*, spanspec*, thead?, tfoot?, tbody)>
<!ATTLIST tgroup
   align       (left|center|right|justify|char) #IMPLIED
   char        CDATA     #IMPLIED
   charoff     NUTOKEN   #IMPLIED
   colsep      %yesorno; #IMPLIED
   cols     NUMBER    #REQUIRED
   rowsep      %yesorno; #IMPLIED
   tgroupstyle NMTOKEN   #IMPLIED
>

<!ELEMENT colspec - O EMPTY>
<!ATTLIST colspec
   align    (left|center|right|justify|char) #IMPLIED
   char     CDATA     #IMPLIED
   charoff  NUTOKEN   #IMPLIED
   colname  NMTOKEN   #IMPLIED
   colnum   NUMBER    #IMPLIED
   colsep   %yesorno; #IMPLIED
   colwidth CDATA     #IMPLIED
   rowsep   %yesorno; #IMPLIED
>

<!ELEMENT spanspec - O EMPTY>
<!ATTLIST spanspec
   align    (left|center|right|justify|char) #IMPLIED
   char     CDATA     #IMPLIED
   charoff  NUTOKEN   #IMPLIED
   colsep   %yesorno; #IMPLIED
   nameend  NMTOKEN   #REQUIRED
   namest   NMTOKEN   #REQUIRED
   rowsep   %yesorno; #IMPLIED
   spanname NMTOKEN   #REQUIRED
>
```

```
<!ELEMENT thead - O (colspec*, row+)>
<!ATTLIST thead
   valign (top|middle|bottom) "bottom"
>

<!ELEMENT tfoot - O (colspec*, row+)>
<!ATTLIST tfoot
   valign (top|middle|bottom) "top"
>

<!ELEMENT tbody - O (row+)>
<!ATTLIST tbody
   valign (top|middle|bottom) "top"
>

<!ELEMENT row - O (entry+)>
<!ATTLIST row
   rowsep %yesorno;           #IMPLIED
   valign (top|middle|bottom) "top"
>

<!ELEMENT entry - O (#PCDATA)>
<!ATTLIST entry
   align    (left|center|right|justify|char) #IMPLIED
   char     CDATA     #IMPLIED
   charoff  NUTOKEN   #IMPLIED
   colname  NMTOKEN   #IMPLIED
   colsep   %yesorno; #IMPLIED
   morerows NUMBER    #IMPLIED
   nameend  NMTOKEN   #IMPLIED
   namest   NMTOKEN   #IMPLIED
   rotate   %yesorno; #IMPLIED
   rowsep   %yesorno; #IMPLIED
   spanname NMTOKEN   #IMPLIED
   valign   (top|middle|bottom) #IMPLIED
>
```

Element structure

A CALS table has an optional title followed by one or more `tgroup` elements. This allows, for example, different portions of one table to have different numbers of columns. In practice, most CALS tables have a single `tgroup` element. The `tgroup` element is the major portion of the table. It has several optional parts: multiple `colspec` and `spanspec` elements followed by (at most) one heading and one footing element. The only required sub-element of a `tgroup` element is its body. Unlike the FrameMaker model of table structure, the CALS model has its `tgroup` element appearing after the footing element.

The colspec empty element has attributes describing characteristics of a table column. The spanspec empty element has attributes describing straddling characteristics of a portion of a table. These elements have no counterpart in FrameMaker. They exist only to have their attribute values specify information about other elements in the table.

The thead and tfoot heading and footing elements contain their own optional colspec elements followed by one or more rows.

The tbody element contains one or more rows.

As supported by FrameMaker, a table row consists of a set of cells in entry elements, each of which can contain only text. Readers familiar with the CALS model may notice that these declarations do not include the entrytbl element which supports creating tables within tables. FrameMaker does not allow tables within tables, so does not support this element.

Attribute structure

Elements in the CALS table model use attributes to describe properties of the table such as cell alignment or straddling behavior.

Inheriting attribute values

Some attributes are common to several elements in the description of a table. In these cases, attribute values are inherited in the element hierarchy. The values of attributes associated with colspec and spanspec elements act as though they were on the parent element for inheritance purposes. That is, if a tgroup element has two colspec child elements and a thead child element, the attributes of the colspec elements apply to the thead element unless that element has its own colspec elements with attribute values that override the inherited ones.

Orient attribute

In the CALS model, the table element has an orient attribute. This attribute is not supported in FrameMaker, because there is no way in a FrameMaker table to specify orientation on the page.

Straddling attributes

A spanspec element describes a column range so that a straddle cell can describe which columns it spans by referencing a spanspec through its spanname attribute.

An entry element specifies which columns it occupies by one of three methods:

- Using the namest and nameend attributes to reference columns explicitly. The namest attribute indicates the first column in the straddle; the nameend attribute indicates the last column.
- Using the spanname attribute as an indirect reference to the columns.

- Using the `colname` attribute (for a non-straddled cell).

8 Read/Write Rules for the CALS/OASIS Table Model

By default, FrameMaker can read and write CALS (or OASIS) tables without your intervention. For information on what it does by default and how you can change that behavior with read/write rules, see Chapter 22, "Translating Tables" in Developer Guide. FrameMaker does not use read/write rules to implement its default interpretation of CALS tables. However, to help your understanding of the default interpretation, this chapter contains a set of rules that encapsulate the software's default behavior for CALS tables.

As described in Chapter 22, "Translating Tables," (in Developer Guide) the software's default behavior is different depending on whether the `table` element is a container element or a table element in FrameMaker. The only difference is what type of element `table` becomes and what happens to the `tgroup` element. All other elements and attributes always translate in the same way.

```
element "table" {
 /* If table is a container element, use this subrule: */
   is fm element;

 /* If table is a table element, use this subrule: */
   is fm table element;

 /* The rest of the subrules for table are always applicable. */
   attribute "tabstyle" is fm property table format;
   attribute "tocentry" is fm attribute;
   attribute "frame"
   {
      is fm property table border ruling;
      value "top"    is fm property value top;
      value "bottom" is fm property value bottom;
      value "topbot" is fm property value top and bottom;
      value "all"    is fm property value all;
      value "sides"  is fm property value sides;
      value "none"   is fm property value none;
   }
   attribute "colsep" is fm property column ruling;
   attribute "rowsep" is fm property row ruling;
   attribute "orient" is fm attribute;
   attribute "pgwide" is fm property page wide;
}
```

```
element "tgroup"
{
   /* If table is a container element, use this subrule: */
   is fm table element;
   /* If table is a table element, use this subrule: */
   unwrap;
   /*The rest of the subrules for tgroup are always applicable.*/
   attribute "cols"        is fm property columns;
   attribute "tgroupstyle" is fm property table format;
   attribute "colsep"      is fm property column ruling;
   attribute "rowsep"      is fm property row ruling;
   attribute "align"       is fm attribute;
   attribute "charoff"     is fm attribute;
   attribute "char"        is fm attribute;
}
element "colspec"
{
   is fm colspec;
   attribute "colnum"   is fm property column number;
   attribute "colname"  is fm property column name;
   attribute "align"    is fm property cell alignment type;
   attribute "charoff"  is fm property cell alignment offset;
   attribute "char"     is fm property cell alignment character;
   attribute "colwidth" is fm property column width;
   attribute "colsep"   is fm property column ruling;
   attribute "rowsep"   is fm property row ruling;
}
element "spanspec"
{
   is fm spanspec;
   attribute "spanname" is fm property span name;
   attribute "namest"   is fm property start column name;
   attribute "nameend"  is fm property end column name;
   attribute "align"    is fm property cell alignment type;
   attribute "charoff"  is fm property cell alignment offset;
   attribute "char"     is fm property cell alignment character;
   attribute "colsep"   is fm property column ruling;
   attribute "rowsep"   is fm property row ruling;
}
```

```
element "thead"
{
   is fm table heading element;
   attribute "valign" is fm attribute;
}

element "tfoot"
{
   is fm table footing element;
   attribute "valign" is fm attribute;
}

element "tbody"
{
   is fm table body element;
   attribute "valign" is fm attribute;
}

element "row"
{
   is fm table row element;
   attribute "valign" is fm attribute;
   attribute "rowsep" is fm property row ruling;
}

element "entry"
{
   is fm table cell element;
   attribute "colname"  is fm property column name;
   attribute "namest"   is fm property start column name;
   attribute "nameend"  is fm property end column name;
   attribute "spanname" is fm property span name;
   attribute "morerows" is fm property more rows;
   attribute "colsep"   is fm property column ruling;
   attribute "rowsep"   is fm property row ruling;
   attribute "rotate"   is fm property rotate;
   attribute "valign"   is fm attribute;
   attribute "align"    is fm attribute;
   attribute "charoff"  is fm attribute;
   attribute "char"     is fm attribute;
}
```

9 SGML Declaration

To be complete, an SGML document must start with an SGML declaration. This chapter contains the text of the SGML declaration used by FrameMaker when you do not supply one. It also describes the variants of the concrete syntax that you can use in your SGML declaration and unsupported optional SGML features.

> **Note: XML:** The XML specification states that XML must use a specific SGML declaration. This chapter pertains only to SGML structure applications. If you are only working with XML markup, you may skip this chapter.

When you import an SGML document, FrameMaker first searches for the declaration in the SGML document. If the software does not find the declaration there, it looks for an SGML declaration specified by your SGML application definition. If your definition does not specify an SGML declaration, then the software uses the declaration described below.

When you export a FrameMaker document to SGML, FrameMaker first tries to use an SGML declaration you specified by your application. If you haven't specified one, it uses the SGML declaration described below.

For information on how to specify an SGML declaration as part of an application, see Developer Guide, page 134: Application definition file.

Text of the default SGML declaration

The SGML declaration provided by FrameMaker uses ISO Latin-1 as the character set, the reference concrete syntax, and the reference capacity set. The declaration enables the optional features `OMITTAG`, `SHORTTAG`, and `FORMAL`.

For information on the default translation between the FrameMaker and ISO Latin-1 character sets, see Chapter 12, "Character Set Mapping." For information on using other ISO character sets, see Chapter 11, "ISO Public Entities."

The text of the default SGML declaration is as follows:

```
<!SGML "ISO 8879:1986"

CHARSET

   BASESET "ISO Registration Number 100//CHARSET ECMA-94 Right
Part of Latin Alphabet Nr. 1//ESC 2/13 4/1"
```

```
   DESCSET
      0 9 UNUSED
      9 2 9
      11 2 UNUSED
      13 1 13
      14 18 UNUSED
      32 95 32
      127 1 UNUSED
      128 127 128
      255 1 UNUSED

CAPACITY
      PUBLIC "ISO 8879:1986//CAPACITY Reference//EN"

SCOPE DOCUMENT

SYNTAX

   SHUNCHAR         0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20
            21 22 23 24 25 26 27 28 29 30 31 127 255

   BASESET "ISO Registration Number 100//CHARSET ECMA-94 Right
Part of Latin Alphabet Nr. 1//ESC 2/13 4/1"

    DESCSET 0 256 0

   FUNCTION RE 13
            RS 10
            SPACE 32
            TAB SEPCHAR 9

   NAMING         LCNMSTRT ""
            UCNMSTRT ""
            LCNMCHAR "&#45;&#46;"
            UCNMCHAR "&#45;&#46;"
            NAMECASE
               GENERAL YES
               ENTITY NO

   DELIM      GENERAL SGMLREF
            SHORTREF SGMLREF

   NAMES SGMLREF

   QUANTITY SGMLREF

FEATURES
```

```
    MINIMIZE DATATAG NO
             OMITTAG YES
             RANK NO
             SHORTTAG YES

    LINK      SIMPLE NO
          IMPLICIT NO
          EXPLICIT NO

    OTHER CONCUR NO
          SUBDOC NO
          FORMAL YES

APPINFO NONE

>
```

SGML concrete syntax variants

The SGML parser used by FrameMaker allows these modifications to the SGML reference concrete syntax:

- The NAMECASE parameter of the SGML declaration can be changed. The default settings below specify that general names are *not* case sensitive (YES), and entity names *are* case sensitive (NO):

  ```
  NAMECASE
        GENERAL  YES
        ENTITY   NO
  ```

- Reserved names can be changed.
- Short references can, but need not, be used. If they are used, the only possible short reference delimiter set is that of the reference concrete syntax.
- The value for the NAMELEN quantity can be increased up to 239.
- The values for the following quantities can be increased, but not to more than 30 times their value in the reference concrete syntax:

  ```
  ATTCNT
  ATTSPLEN
  BSEQLEN
  ENTLVL
  LITLEN
  PILEN
  TAGLEN
  TAGLVL
  ```

- The following quantities can be increased up to 253:

  ```
  GRPCNT
  GRPGTCNT
  GRPLVL
  ```

No SGML read/write rules are needed to provide for variant concrete syntaxes. FrameMaker obtains the information from the SGML declaration.

The concrete syntax declared in the SGML declaration must be used for the entire document; if a variant concrete syntax is declared, the reference concrete syntax cannot be used in the prolog. Thus, the concrete syntax scope parameter must be:

```
SCOPE DOCUMENT
```

Unsupported optional SGML features

The SGML standard defines some features as optional, meaning that a specific implementation does not have to accommodate these features to be considered a conforming SGML system.

The following optional SGML features are not supported by FrameMaker:

- `DATATAG`
- `RANK`
- `LINK`
- `SUBDOC`
- `CONCUR`

Your DTD and SGML documents cannot use any of these features. If they do, the FrameMaker signals an error and terminates processing. You cannot change this behavior by providing an SGML API client.

10 ISO Public Entities

Annex D of the SGML standard defines several sets of internal SDATA entities. Each entity represents a character; each entity set is a logical grouping of these entities. DTDs frequently include these entity sets by using parameter entity references to external entities accessed with a public identifier. People in the SGML community frequently interchange DTDs and SGML documents with such entity references and assume that the recipient can interpret the public identifiers. FrameMaker includes copies of these entity sets and makes them available using the default handling of public identifiers.

> **Note: XML:** The XML specification does not allow SDATA entities, but it does allow UNICODE and predefined character entities for special characters. This chapter pertains only to SGML structure applications. If you are only working with XML markup, you may skip this chapter.

These entity sets are defined in an ISO standard and are accessed with public identifiers, so they are commonly known as *ISO public entity sets*. The public entity sets fall into the following categories:

Entity set	Description
Latin alphabetic characters	Latin alphabetic characters used in Western European languages
Greek alphabetic characters	Letters of the Greek alphabet
Greek symbols	Greek character names for use as variable names in technical applications
Cyrillic alphabetic characters	Cyrillic characters used in the Russian language
Numeric and special graphic characters	Minimum data characters and reference concrete syntax characters
Diacritical mark characters	Diacritical marks
Publishing characters	Well-known publishing characters
Technical symbols	Technical symbols
Added math symbols	Mathematical symbols

If your application uses FrameMaker's support of ISO entity sets, you may want to create palettes your end user can use to enter these entities in a FrameMaker document. For information on creating these palettes, see Developer Guide, Facilitating entry of special characters that translate as entities.

What you need to use ISO public entities

For your end users to use characters from the ISO public entity sets, your application needs two pieces of information for each character entity: the entity's declaration, and an SGML read/write rule that tells FrameMaker how to translate a reference to that entity in an SGML document to a character or variable in a FrameMaker document. FrameMaker provides this information in two files for each entity set.

All files used for ISO public entity sets are in the directory `$STRUCTDIR/isoents`. For information on the location of this directory on your system, see Developer Guide, page 131: Location of structure files. The files for each entity set are as follows:

Entity set	Entity declaration files	Read/write rules files
Latin alphabetic characters	`isolat1.ent` `isolat2.ent`	`isolat1.rw` `isolat2.rw`
Greek alphabetic characters	`isogrk1.ent` `isogrk2.ent`	`isogrk1.rw` `isogrk2.rw`
Greek symbols	`isogrk3.ent` `isogrk4.ent`	`isogrk3.rw` `isogrk4.rw`
Cyrillic alphabetic characters	`isocyr1.ent` `isocyr2.ent`	`isocyr1.rw` `isocyr2.rw`
Numeric and special graphic characters	`isonum.ent`	`isonum.rw`
Diacritical mark characters	`isodia.ent`	`isodia.rw`
Publishing characters	`isopub.ent`	`isopub.rw`
Technical symbols	`isobox.ent` `isotech.ent`	`isobox.rw` `isotech.rw`

Added math symbols	`isoamso.ent`
	`isoamsb.ent`
	`isoamsr.ent`
	`isoamsn.ent`
	`isoamsa.ent`
	`isoamsc.ent`
		`isoamso.rw`
		`isoamsb.rw`
		`isoamsr.rw`
		`isoamsn.rw`
		`isoamsa.rw`
		`isoamsc.rw`

Entity declaration files

Each entity declaration file starts with two comment declarations that suggest both the public identifier and the entity name by which to identify the entity set. For the ISO Latin-1 entity set, these comments are:

```
<!-- (C) International Organization for Standardization 1986
      Permission to copy in any form is granted for use with
      conforming SGML systems and applications as defined in
      ISO 8879, provided this notice is included in all copies.
-->
<!-- Character entity set. Typical invocation:
      <!ENTITY % ISOlat1 PUBLIC
         "ISO 8879-1986//ENTITIES Added Latin 1//EN">
      %ISOlat1;
-->
```

After the initial comments, an entity declaration file consists of a sequence of entity declarations. For example, the first few entity declarations for ISO Latin-1 are as follows:

```
<!ENTITY aacute SDATA "[aacute]"--=small a, acute accent-->
<!ENTITY Aacute SDATA "[Aacute]"--=capital A, acute accent-->
<!ENTITY acirc SDATA "[acirc ]"--=small a, circumflex accent-->
<!ENTITY Acirc SDATA "[Acirc ]"--=capital A, circumflex accent-->
<!ENTITY agrave SDATA "[agrave]"--=small a, grave accent-->
<!ENTITY Agrave SDATA "[Agrave]"--=capital A, grave accent-->
<!ENTITY aring SDATA "[aring ]"--=small a, ring-->
<!ENTITY Aring SDATA "[Aring ]"--=capital A, ring-->
```

You should never modify these files, because they provide the standard ISO public entity declarations.

If your SGML documents use the standard invocations for ISO public entity sets, you do not have to provide any information in your application definition on where to find these entities; FrameMaker finds them in the default directory. If necessary, you can provide explicit `public` statements to substitute alternative versions of the entity sets. For information on working with application definitions, see Developer Guide, page 134: Application definition file.

Entity read/write rules files

FrameMaker provides read/write rules for many of the entities in the ISO public entity sets. The rules are organized in files for each public entity set. These files are not complete rules documents. Instead, they are simply lists of rules or comments explaining which entities do not have default correspondences.

You can include individual files in your application's read/write rules document by using the `#include` statement. To include the rules for all of the ISO public entity sets, use this single statement:

```
#include isoall.rw
```

To include only the ISO Latin-1 entity set, use these statements:

```
#include isolat1.rw
#include isolat2.rw
```

For more information on read/write rules files, see Developer Guide, Chapter 18, Read/Write Rules and Their Syntax

Format of entity rules

By default, FrameMaker has rules for each entity that can be represented in FrameMaker using the standard FrameMaker character set, the Symbol font, or the Zapf Dingbat font and for a few (such as the fractions in `isonum`) that can be represented with a FrameMaker user variable. Entities that cannot be represented in this way do not have a default translation. Users of your application may have more fonts available. If so, you can modify these rules files to include translations for other entities.

The default rules for entities available in the default character sets or through variables differ depending on how FrameMaker translates the entity.

- If the character appears in FrameMaker's standard character set and requires no special formatting, the rule has the following form:

  ```
  entity "ename" is fm char code;
  ```

  where *ename* is the entity name and *code* is the character code. For example, the following rule is for the small letter "a" with an acute accent:

  ```
  entity "aacute" is fm char 0x87;
  ```

- If the character appears in FrameMaker's Symbol or Zapf Dingbat character set or appears in FrameMaker's standard character set, but requires special formatting, the rule has the following form:

  ```
  entity "ename" is fm char code in "fmtag";
  ```

  where *ename* is the entity name, *code* is the character code, and *fmtag* is one of the character tags defined below. For example, the following rule is for the plus-or-minus sign:

  ```
  entity "plusnm" is fm char 0xb1 in "FmSymbol";
  ```

- If the character can be represented by an FrameMaker variable, the rule has the following form:

  ```
  entity "ename" is fm variable "var";
  ```

  where *ename* is the entity name and *var* is one of the FrameMaker variables defined below. For example, the following rule is for the fraction one-half:

  ```
  entity "frac12" is fm variable "FmFrac12";
  ```

For details on how each entity translates into a FrameMaker document, see the individual rules files.

Character formats

As mentioned above, the rules for some character entities refer to FrameMaker character formats or variable names. FrameMaker has default definitions for these character formats:

Character format	Defined as
FmDenominator	Default font, subscripted; other characteristics As Is
FmDingbats	Zapf Dingbat font; other characteristics As Is
FmNumerator	Default font, superscripted; other characteristics As Is
FmSdata	Default font, underlined and in green; other characteristics As Is
FmSuperscript	Default font superscripted; other characteristics As Is
FmSymbol	Symbol font; other characteristics As Is
FmUnderlineSymbol	Symbol font, underlined; other characteristics As Is

Variables

FrameMaker also has default definitions for these variables:

Variable	Defined as
FmCare-of	c/o
FmEmsp13	an em space
FmFrac12	1/2
FmFrac13	1/3
FmFrac14	1/4
FmFrac15	1/5
FmFrac16	1/6
FmFrac18	1/8
FmFrac23	2/3
FmFrac25	2/5
FmFrac34	3/4
FmFrac35	3/5
FmFrac38	3/8
FmFrac45	4/5
FmFrac56	5/6
FmFrac58	5/8
FmFrac78	7/8

Your end user's documents may not have these character formats or variables defined. When FrameMaker imports an SGML document with an entity reference that needs one of these formats or variables, it checks whether the template defined in the SGML application provides the definition. If so, it uses the information from the template. If not, it uses its own definitions, copying the definition to the appropriate catalog of the document being processed and using it to process the entity.

What happens with the declarations and rules

Your application may use some or all of the entity declarations and read/write rules provided with FrameMaker. In addition, you may choose to have different declarations or rules for some or all of the entities.

If you want to use the translations provided by FrameMaker with no changes, you do so in one of two ways.

- If your application has no other read/write rules, then you do not have to explicitly mention the rules for these entity sets. That is, if the definition of your application does not include a read/write rules file, FrameMaker behaves as though it had a rules file that included only the ISO public entity rules.
- On the other hand, if your application does have a read/write rules file, then that file must explicitly include the rules for the ISO public entity sets in which you're interested. If you want all of them, add the following line to your file:

  ```
  #include isoall.rw
  ```

  When you create a new read/write rules file, this line is automatically included.

If you want to use only the rules that FrameMaker provides, be sure that your rules file has no additional `entity` rules referring to these entities. However, you may want to have FrameMaker translate most but not all of these entities in the way it provides, while you change the behavior for some of them with rules or entity declarations. To do this, include an extra entity declaration or rule for the appropriate entities.

For example, assume the DTD for your application is called `myapp.dtd` and includes the following lines:

```
<!ENTITY % ISOlat1 PUBLIC
         "ISO 8879-1986//ENTITIES Added Latin 1//EN">
%ISOlat1;
```

Further, assume the application has no rules or has a rules document that contains the following lines:

```
#include "isolat1.rw"
#include "isolat2.rw"
```

The default version of isolat1.rw contains the rule:

```
entity "aacute" is fm char 0x87;
```

This translates references to the aacute entity as the small letter a with an acute accent. Suppose, however, that your application needs this entity, instead, to translate as a particular bitmap that you store as a reference element in the FrameMaker document template. You can accomplish this by adding either a new entity declaration or a new rule.

To continue the example, assume you import an SGML document that begins as follows:

```
<!DOCTYPE myapp SYSTEM "myapp.dtd" [
   <!ENTITY aacute SDATA "fm ref: acute-a">
]>
```

This SGML document has two declarations for aacute. The parser uses the first one it encounters. Since the parser processes the external DTD subset after it processes the internal DTD subset, it finds the declaration that uses the reference element first and this is the entity declaration FrameMaker uses. Since FrameMaker recognizes the fm ref in the parameter literal, it uses that parameter literal to process the entity reference and ignores any rules that refer to the entity. The resulting document includes the reference element for the entity reference.

Instead of including the declaration for aacute that uses the fm ref parameter literal, you can add the following rule to your rules file:

```
entity "aacute" is fm reference element "acute-a";
```

This translates references to the aacute entity as the small letter a with an acute accent. Suppose, however, that your application needs this entity, instead, to translate as a particular bitmap that you store as a reference element in the FrameMaker document template. You accomplish this by adding a rule for that entity before the #include statements, as follows:

```
entity "aacute" is fm reference element "acute-a";
```

Remember that FrameMaker uses the first rule in a rules file that applies to a particular situation. Therefore, if you use this rule, then the line in the example that includes isolat1.rw must occur after this rule. That is, your rules file must look like:

```
entity "aacute" is fm reference element "acute-a";
. . .
#include isolat1.rw
. . .
```

If, instead, it looks like:

```
#include isolat1.rw
. . .
entity "aacute" is fm reference element "acute-a";
. . .
```

FrameMaker finds the rule in isolat1.rw before your rule and use that to process references to the aacute entity.

FrameMaker has rules for entities in the ISO public entity sets for which there is a direct correspondence in one of its standard character sets or which can be created using a character from those character sets. It does not provide rules for entities that would require a different character set or a graphic.

If you reference an ISO public entity for which there is not a rule, the software treats it as it does any other entity that does not have a corresponding rule. You can change this behavior with the `entity` rule. For more information on FrameMaker's translation of entities in the absence of rules and for information on how you can modify this, see Developer Guide, Chapter 21, Translating Entities and Processing Instructions

11 Character Set Mapping

FrameMaker writes SGML documents using the ISO Latin-1 character set. This character set differs from FrameMaker's character set. Consequently, the software uses a default character set mapping to translate between the character sets.

> **Note: XML:** The XML specification allows UNICODE in content and in markup tokens, so the use of ISO character sets is not necessary. FrameMaker supports the full range of UNICODE in the content of an XML document, and offers limited support of characters in markup tokens. For more information, see Developer Guide, Supported characters in element and attribute names.
>
> If you are only working with XML markup, you may skip this chapter.

FrameMaker includes copies of other ISO public entity sets and provides rules to handle them for your application. For information on how FrameMaker supports ISO public entities, see Chapter 11, "ISO Public Entities."

This chapter describes the default mapping between the FrameMaker character set and the ISO Latin-1 character set. You can change this mapping by using the `character map` rule as described in "character map" on page 49.

To determine the mapping for a particular character, use the table on the next page as follows:

- For a character in the ISO Latin-1 character set, find the hexadecimal character code for the character of interest in the leftmost column. Read the mapping in the column labelled "Mapping from ISO Latin-1 to FrameMaker." The entry on the left side of the equal sign is the ISO Latin-1 character code. The entry on the right side of the equal sign is the character's translation in FrameMaker. For example, the character code `\xA1` has the entry:

  ```
  \xA1 = \xC1
  ```

  This means that the ISO Latin-1 character `\xA1` translates to the FrameMaker character `\xC1`.

- For a character in the FrameMaker character set, find the hexadecimal character code for the character of interest in the leftmost column. Read the mapping in the column labelled "Mapping from FrameMaker to ISO Latin-1." The entry on the right side of the equal sign is the FrameMaker character code. The entry on the left side of the equal sign is the character's translation in ISO Latin-1. For example, the character code `\x10` has the entry:

  ```
  \x20 = \x10
  ```

  This means that the FrameMaker character `\x10` translates to the ISO Latin-1 character `\x20`.

- If there is no row corresponding to a character code, then that character code is the same in both character sets.

Character code	Mapping from ISO Latin-1 to FrameMaker	Mapping from FrameMaker to ISO Latin-1
\x00	\x00 = trap	trap = \x00
\x01	\x01 = trap	trap = \x01
\x02	\x02 = trap	trap = \x02
\x03	\x03 = trap	trap = \x03
\x04	\x04 = trap	trap = \x04
\x05	\x05 = trap	trap = \x05
\x06	\x06 = trap	trap = \x06
\x07	\x07 = trap	trap = \x07
\x08	\x08 = trap	\x09 = \x08
\x09	\x09 = \x08	\x0A = \x09
\x0A	\x0A = \x0A	\x0A = \x0A
\x0B	\x0B = trap	trap = \x0B
\x0C	\x0C = trap	trap = \x0C
\x0D	\x0D = trap	trap = \x0D
\x0E	\x0E = trap	trap = \x0E
\x0F	\x0F = trap	trap = \x0F
\x10	\x10 = trap	\x20 = \x10
\x11	\x11 = trap	\x20 = \x11
\x12	\x12 = trap	\x20 = \x12
\x13	\x13 = trap	\x20 = \x13
\x14	\x14 = trap	\x20 = \x14
\x15	\x15 = trap	\x2D = \x15
\x16	\x16 = trap	trap = \x16
\x17	\x17 = trap	trap = \x17
\x18	\x18 = trap	trap = \x18
\x19	\x19 = trap	trap = \x19
\x1A	\x1A = trap	trap = \x1A
\x1B	\x1B = trap	trap = \x1B
\x1C	\x1C = trap	trap = \x1C
\x1D	\x1D = trap	trap = \x1D
\x1E	\x1E = trap	trap = \x1E
\x1F	\x1F = trap	trap = \x1F

Character code	Mapping from ISO Latin-1 to FrameMaker	Mapping from FrameMaker to ISO Latin-1
\x7F	\x7F = trap	trap = \x7F
\x80	\x80 = trap	\xC4 = \x80
\x81	\x81 = trap	\xC5 = \x81
\x82	\x82 = trap	\xC7 = \x82
\x83	\x83 = trap	\xC9 = \x83
\x84	\x84 = trap	\xD1 = \x84
\x85	\x85 = trap	\xD6 = \x85
\x86	\x86 = trap	\xDC = \x86
\x87	\x87 = trap	\xE1 = \x87
\x88	\x88 = trap	\xE0 = \x88
\x89	\x89 = trap	\xE2 = \x89
\x8A	\x8A = trap	\xE4 = \x8A
\x8B	\x8B = trap	\xE3 = \x8B
\x8C	\x8C = trap	\xE5 = \x8C
\x8D	\x8D = trap	\xE7 = \x8D
\x8E	\x8E = trap	\xE9 = \x8E
\x8F	\x8F = trap	\xE8 = \x8F
\x90	\x90 = trap	\xEA = \x90
\x91	\x91 = trap	\xEB = \x91
\x92	\x92 = trap	\xED = \x92
\x93	\x93 = trap	\xEC = \x93
\x94	\x94 = trap	\xEE = \x94
\x95	\x95 = trap	\xEF = \x95
\x96	\x96 = trap	\xF1 = \x96
\x97	\x97 = trap	\xF3 = \x97
\x98	\x98 = trap	\xF2 = \x98
\x99	\x99 = trap	\xF4 = \x99
\x9A	\x9A = trap	\xF6 = \x9A
\x9B	\x9B = trap	\xF5 = \x9B
\x9C	\x9C = trap	\xFA = \x9C
\x9D	\x9D = trap	\xF9 = \x9D

Character code	Mapping from ISO Latin-1 to FrameMaker	Mapping from FrameMaker to ISO Latin-1
\x9E	\x9E = trap	\xFB = \x9E
\x9F	\x9F = trap	\xFC = \x9F
\xA0	\xA0 = trap	trap = \xA0
\xA1	\xA1 = \xC1	trap = \xA1
\xA2	\xA2 = \xA2	\xA2 = \xA2
\xA3	\xA3 = \xA3	\xA3 = \xA3
\xA4	\xA4 = \xDB	\xA7 = \xA4
\xA5	\xA5 = \xB4	\xB7 = \xA5
\xA6	\xA6 = \x7C	\xB6 = \xA6
\xA7	\xA7 = \xA4	\xDF = \xA7
\xA8	\xA8 = \xAC	\xAE = \xA8
\xA9	\xA9 = \xA9	\xA9 = \xA9
\xAA	\xAA = \xBB	trap = \xAA
\xAB	\xAB = \xC7	\xB4 = \xAB
\xAC	\xAC = \xC2	\xA8 = \xAC
\xAD	\xAD = \x2D	trap = \xAD
\xAE	\xAE = \xA8	\xC6 = \xAE
\xAF	\xAF = \xF8	\xD8 = \xAF
\xB0	\xB0 = \xFB	trap = \xB0
\xB1	\xB1 = trap	trap = \xB1
\xB2	\xB2 = trap	trap = \xB2
\xB3	\xB3 = trap	trap = \xB3
\xB4	\xB4 = \xAB	\xA5 = \xB4
\xB5	\xB5 = trap	trap = \xB5
\xB6	\xB6 = \xA6	trap = \xB6
\xB7	\xB7 = \xA5	trap = \xB7
\xB8	\xB8 = \xFC	trap = \xB8
\xB9	\xB9 = trap	trap = \xB9
\xBA	\xBA = \xBC	trap = \xBA
\xBB	\xBB = \xC8	\xAA = \xBB
\xBC	\xBC = trap	\xBA = \xBC
\xBD	\xBD = trap	trap = \xBD

Character code	Mapping from ISO Latin-1 to FrameMaker	Mapping from FrameMaker to ISO Latin-1
\xBE	\xBE = trap	\xE6 = \xBE
\xBF	\xBF = \xC0	\xF8 = \xBF
\xC0	\xC0 = \xCB	\xBF = \xC0
\xC1	\xC1 = \xE7	\xA1 = \xC1
\xC2	\xC2 = \xE5	\xAC = \xC2
\xC3	\xC3 = \xCC	trap = \xC3
\xC4	\xC4 = \x80	trap = \xC4
\xC5	\xC5 = \x81	trap = \xC5
\xC6	\xC6 = \xAE	trap = \xC6
\xC7	\xC7 = \x82	\xAB = \xC7
\xC8	\xC8 = \xE9	\xBB = \xC8
\xC9	\xC9 = \x83	trap = \xC9
\xCA	\xCA = \xE6	trap = \xCA
\xCB	\xCB = \xE8	\xC0 = \xCB
\xCC	\xCC = \xED	\xC3 = \xCC
\xCD	\xCD = \xEA	\xD5 = \xCD
\xCE	\xCE = \xEB	trap = \xCE
\xCF	\xCF = \xEC	trap = \xCF
\xD0	\xD0 = trap	\x2D = \xD0
\xD1	\xD1 = \x84	\x2D = \xD1
\xD2	\xD2 = \xF1	\x22 = \xD2
\xD3	\xD3 = \xEE	\x22 = \xD3
\xD4	\xD4 = \xEF	\x60 = \xD4
\xD5	\xD5 = \xCD	\x27 = \xD5
\xD6	\xD6 = \x85	trap = \xD6
\xD7	\xD7 = trap	trap = \xD7
\xD8	\xD8 = \xAF	\xFF = \xD8
\xD9	\xD9 = \xF4	trap = \xD9
\xDA	\xDA = \xF2	\x2F = \xDA
\xDB	\xDB = \xF3	\xA4 = \xDB
\xDC	\xDC = \x86	trap = \xDC
\xDD	\xDD = trap	trap = \xDD

Character code	Mapping from ISO Latin-1 to FrameMaker	Mapping from FrameMaker to ISO Latin-1
\xDE	\xDE = trap	trap = \xDE
\xDF	\xDF = \xA7	trap = \xDF
\xE0	\xE0 = \x88	trap = \xE0
\xE1	\xE1 = \x87	\xB7 = \xE1
\xE2	\xE2 = \x89	\x2C = \xE2
\xE3	\xE3 = \x8B	trap = \xE3
\xE4	\xE4 = \x8A	trap = \xE4
\xE5	\xE5 = \x8C	\xC2 = \xE5
\xE6	\xE6 = \xBE	\xCA = \xE6
\xE7	\xE7 = \x8D	\xC1 = \xE7
\xE8	\xE8 = \x8F	\xCB = \xE8
\xE9	\xE9 = \x8E	\xC8 = \xE9
\xEA	\xEA = \x90	\xCD = \xEA
\xEB	\xEB = \x91	\xCE = \xEB
\xEC	\xEC = \x93	\xCF = \xEC
\xED	\xED = \x92	\xCC = \xED
\xEE	\xEE = \x94	\xD3 = \xEE
\xEF	\xEF = \x95	\xD4 = \xEF
\xF0	\xF0 = trap	trap = \xF0
\xF1	\xF1 = \x96	\xD2 = \xF1
\xF2	\xF2 = \x98	\xDA = \xF2
\xF3	\xF3 = \x97	\xDB = \xF3
\xF4	\xF4 = \x99	\xD9 = \xF4
\xF5	\xF5 = \x9B	trap = \xF5
\xF6	\xF6 = \x9A	\x5E = \xF6
\xF7	\xF7 = trap	\x7E = \xF7
\xF8	\xF8 = \xBF	\xAF = \xF8
\xF9	\xF9 = \x9D	trap = \xF9
\xFA	\xFA = \x9C	trap = \xFA
\xFB	\xFB = \x9E	\xB0 = \xFB
\xFC	\xFC = \x9F	\xB8 = \xFC
\xFD	\xFD = trap	trap = \xFD

Character code	Mapping from ISO Latin-1 to FrameMaker	Mapping from FrameMaker to ISO Latin-1
\xFE	\xFE = trap	trap = \xFE
\xFF	\xFF = \xD8	trap = \xFF

Glossary

This glossary contains common terms used by FrameMaker, XML, and SGML. For references to more information about the terms, see the index.

ancestor An element that contains a given element in a document's structure. For example, if a `Section` element contains a `Head` element followed by a `Paragraph` element, and the `Paragraph` contains a `Variable` element, the `Paragraph` and `Section` elements are both ancestors of the `Variable` element, but the `Head` element is not an ancestor of the `Variable` element. *See also* descendant, child element, parent element, and sibling.

API Application Programming Interface. Enables developers to create API clients with other applications, such as databases, document management systems, CAD tools, and user interfaces, for automation, database publishing, HTML conversion and other purposes.

application definition A data structure (and the associated files) describing part of a complete XML or SGML application assembled with FrameMaker. You store application definitions in the `structapps.fm` file.

attribute A place to supply information about an element other than its hierarchical position and structure. An attribute value does not add content to a document.

attribute definition The construct used to define a single attribute in a FrameMaker EDD or a DTD.

attribute definition list declaration In markup, the declaration that provides the list of attribute definitions for one or more elements. Also called an ATTLIST. *See also* element declaration.

book A grouping of FrameMaker documents that lets you work with them as a single unit. Lets you generate a single table of contents or other file from the documents, and simplifies printing, numbering, cross-referencing, and formatting.

CALS Continuous Acquisition and Life Cycle Support. The US Department of Defense standard for the electronic delivery of documents.

catalog A floating palette that stores predefined paragraph, character, or table formats.

CDATA In markup, character data. In character data, no markup is recognized, other than the delimiters that end the character data. *See also* NDATA, #PCDATA, RCDATA, and SDATA.

child element An element that is contained in a given element and that is one level below the given element. For example, if a `Section` element contains a `Head` element followed by a `Paragraph` element, and the `Paragraph` element contains a

	`Variable` element, the `Head` and `Paragraph` elements are both child elements of the `Section` element, but the `Variable` element is not. *See also* parent element, ancestor, descendant, and sibling.
concrete syntax	In SGML, a set of choices on the markup a document will use. Since SGML does not require any particular values for these choices, an SGML document requires a concrete syntax so a parser can correctly interpret it. *See also* reference concrete syntax.
container element	In FrameMaker, an element that can contain text, other elements, or both. Contrasts with certain specific element types—for example, a cross-reference element, which can contain nothing other than the cross-reference.
content model	In markup, the part of an element declaration that specifies both a model group and exceptions that define the allowed content of the element. Each markup element declaration has either a content model or declared content. *See also* content rules, declared content, general rule, and model group.
content rules	In FrameMaker, the part of an element declaration that specifies both the element's type and the kind of contents the element can have. *See also* format rules, content model, and general rule.
conversion table	In FrameMaker, a table associating parts of an unstructured document with their structured counterparts, used in converting an unstructured document to a structured document.
cross-reference	A passage in one place in a document that refers to another place, its cross-reference source, in the same or a different document.
cross-reference source	The place referred to by a cross-reference.
data	In markup, the characters of a document that represent the inherent information content. Such characters are not recognized as markup. *See also* markup.
data content notation	In markup, an application-specific interpretation of an element's data content, or of a data entity, that usually extends or differs from the normal meaning of the document character set. Frequently used to identify the format of an external entity containing a graphic.
declaration	In markup, markup that controls how other markup of a document is to be interpreted.
declared content	In an markup element declaration, specifies that the defined element's content is one of the reserved types `CDATA`, `RCDATA`, or `EMPTY`.
declared value	In an markup attribute definition, determines the type of attribute value, such as `ID` or `NUTOKEN`, that is valid when the attribute is specified. Although markup does not define the term *attribute type*, you can loosely think of an attribute's declared value as its type.

default value In markup, the portion of an attribute definition that indicates whether an attribute is required and what value to use if the user does not specify one. In FrameMaker, refers only to the value to use if a user does not supply a value for an attribute.

delimiter In markup, a character string used to identify a piece of markup or to distinguish markup from data. For example, > (greater-than sign) is the default closing delimiter for element tags.

descendant Any element that is below a given element in a document's structure. For example, if a `Section` element contains a `Head` element followed by a `Paragraph` element, and the `Paragraph` element contains a `Variable` element, the `Variable` element is a descendant of both the `Paragraph` and the `Section` elements, but not of the `Head` element. *See also* ancestor, child element, parent element, and sibling.

DOCTYPE In markup, the reserved name that follows the opening delimiter of a DTD. Informally used to refer to the document element.

document A collection of information that is processed as a unit. A FrameMaker document is any file in FrameMaker format. A markup document includes an SGML declaration (for SGML), prologue, and document instance set.

document element In markup, the highest-level element in a document. The generic identifier of this element is specified immediately after the `DOCTYPE` reserved name in the DTD.

document instance In markup, the portion of a document that contains markup and data for a particular project such as a memo or book.

document type A class of documents having similar characteristics, such as technical manual or internal memo.

document type declaration In markup, a document type declaration (DTD) associates a document element with a set of declarations (the document type declaration subset).

document type declaration subset In markup, a set of declarations determining such things as the markup to allow in a document and the elements and attributes for a document set. *See also* external DTD subset and internal DTD subset.

DTD *See* document type declaration subset.

EDD *See* element definition document.

element A structural unit of a document. Holds and organizes the contents of the document.

Element Catalog In FrameMaker, the information extracted from an EDD and stored within each structured FrameMaker document. Makes an external element definition document unnecessary. *See also* element definition document.

element declaration In markup, information describing a particular element. Includes both a name (generic identifier) for the element and content rules. A markup document has an element declaration for each allowed element.

element definition In FrameMaker, a set of rules describing an element. Includes a name (tag) for the element, content rules, and (optionally) context-sensitive format rules. A structured document has an element definition for each element allowed. *See also* content rules and format rules.

element definition document A FrameMaker document that contains a set of element definitions for a class of documents. Can also include information on system defaults and on a structure application with which to associate this information. Also called an EDD.

element tag In FrameMaker, the name assigned to an element and stored in the Element Catalog. *See also* generic identifier.

EMPTY Keyword in an element definition indicating that the element cannot have content. In markup, `EMPTY` is a declared content.

end-tag In markup, the markup that indicates the end of an element.

entity In markup, a collection of characters that can be referenced as a unit. Used for many purposes in markup, such as graphics or frequently used sets of characters.

exclusion An exception to the general rule or content model of an element. Specifies other elements that cannot appear anywhere in the element or in its descendants. Exclusions are not allowed in XML.

external cross-reference In FrameMaker, a cross-reference to a source in a different file. Markup does not define this concept.

external DTD subset In markup, an informal term for an external entity for which an external identifier appears at the beginning of a document type declaration and that is automatically referenced at the end of the document type declaration subset.

external entity In markup, an entity that specifies an external object, such as a file.

facet A pictorial representation of graphical data.

FDK client In FrameMaker, any application created using the Frame Developer's Kit. *See also* Structure API client.

flow *See* , "text flow."

format rules In FrameMaker, the part of an element definition that specifies which predefined format to apply to an element. Format rules can use different formats for different contexts in a document. *See also* content rules.

general entity In markup, an entity that can be referenced from within the content of an element or an attribute value literal.

general rule In FrameMaker, a rule that specifies valid contents for an element and the order in which the contents can appear. Equivalent to the declared content of an

element or the model group part of the content model of an element in markup. *See also* content rules.

generic identifier In markup, the name identifying an element. *See also* element definition and element tag.

highest-level rule In FrameMaker, a read/write rule that is not a subrule of another read/write rule.

HTML Hypertext Markup Language. An encoding system used to describe the content and organization of an electronic document published on the World Wide Web.

ID attribute An attribute of type `ID`, frequently used as an identifier to mark the source of a cross-reference. In a single document, a particular value for an `ID` attribute can be used only once.

IDREF attribute An attribute whose value must be that of an `ID` attribute in the same markup document or FrameMaker document or book. Frequently used for cross-references.

impliable attribute In markup, an attribute whose value does not have to be supplied. If a document does not supply a value, it is up to the processing software to correctly interpret the attribute. Such attributes use the default value `#IMPLIED`.

inclusion An exception to the general rule or content model of an element. Specifies other elements that can appear anywhere in the element or in its descendants. Inclusions are not allowed in XML DTDs

invalid element An element with contents that do not conform to content rules. May be missing required child elements, may not have a definition in the EDD or DTD, or may have text or child elements in a position not allowed by its content rules or by the exclusion and inclusion rules of its ancestors.

internal cross-reference In FrameMaker, a cross-reference to a source in the same file.

internal DTD subset In markup, an informal term for the declarations in a document type declaration that occur within brackets (dso and dsc delimiters) in the markup document entity, rather than being in an external entity.

internal entity In markup, an entity whose replacement text is determined solely by information in its declaration.

ISO public entity In SGML, an entity that occurs in one of the entity sets defined in Annex D of the SGML Standard. These entities provide commonly used special characters.

marker In FrameMaker, a nonprinting character an end user inserts (such as an index entry) to indicate various types of information.

markup Text added to the data of a document in order to convey information about it, such as hierarchical structure or formatting. This document also uses *markup* to generally refer to XML and SGML.

markup minimization In SGML, any of various conventions for omitting markup in a document, including shortening or omitting tags and shortening entity references.

model group In markup, an ordered list that specifies valid contents for an element (such as child elements) and the order in which the contents can appear. A model group is similar to a FrameMaker general rule.

NAMECASE parameter In SGML, the part of the SGML declaration that determines case-sensitivity of markup.

NDATA In SGML (and implicitly XML), non-SGML data. `NDATA` is data that needs special processing by the markup application. `NDATA` is typically used, for example, when representing graphics—in XML the graphic data would be non-parsed data. *See also* CDATA, #PCDATA, RCDATA, and SDATA.

parameter entity In markup, an entity that can be referenced only within a DTD.

parent element An element that contains a given element and is one level above it in the hierarchy. For example, if a `Section` element contains a `Head` element followed by a `Paragraph` element, the `Section` element is the parent element of the `Head` and `Paragraph` elements, but not of the `Variable` element. *See also* child element, ancestor, descendant, and sibling.

parser *See* validating parser.

#PCDATA In markup, parsed character data. This is normal text that can include markup to be parsed. Occurs in an markup element's model group and corresponds to `<TEXT>` in a FrameMaker element's general rule. *See also* CDATA, NDATA, RCDATA, and SDATA.

prefix Text that is automatically placed before the content of an element. In FrameMaker, defined as part of the formatting of an element. For example, a `Quote` text range element might have an open quotation mark as its prefix and a close quotation mark as its suffix. *See also* suffix.

processing instruction In an markup document, a way of indicating that the application needs to perform some special processing. For example, you can use a processing instruction to indicate a location in an markup document that should have a page break.

public identifier In markup, a way of identifying an external entity. Formal public identifiers have a specified syntax that includes an identifier of the owner of the entity and an indication of the markup construct it provides. Formal public identifiers are typically available to any user of markup, not just the users at a particular company. Informal public identifiers may be available more widely than a single document or system, but perhaps no more widely than within a single company. *See also* system identifier

RCDATA In markup, replaceable character data. In replaceable character data, no markup is recognized, other than character and entity references. RCDATA is valid only in SGML. *See also* CDATA, NDATA, #PCDATA, and SDATA.

read/write rule In FrameMaker, interpreted commands you supply to modify how the software translates between FrameMaker and markup documents.

reference concrete syntax In SGML, a particular concrete syntax defined by the SGML standard. *See also* concrete syntax.

reference page An underlying page that stores repeatedly-used graphics and formatting information.

Rubi text Small characters that appear above Japanese-language characters to indicate pronunciation.

rule *See* SGML read/write rule.

SDATA In SGML, specific character data. One common use is for specific characters that might not be in the standard character set. *See also* CDATA, NDATA, #PCDATA, and RCDATA.

SGML An acronym for Standard Generalized Markup Language.

SGML application Rules that apply SGML to a text processing application. Includes a formal specification of the markup constructs used in the application, expressed in SGML. Can also include non-SGML definitions of semantics, application, conventions, and processing.

SGML declaration In SGML, the part of a document that tells a parser how to interpret markup in the document.

SGML read/write rule *See* read/write rule.

sibling Elements at the same level in the structure and with the same parent element. For example, if a `Section` element contains a `Head` element followed by a `Paragraph` element, the `Head` and `Paragraph` elements are siblings. *See also* ancestor, descendant, child element, and parent element.

source *See* cross-reference source.

start-tag In markup, the markup that indicates the beginning of an element.

Structure API client In FrameMaker, an FDK client created to change the translation between FrameMaker and markup documents. *See also* FDK client.

subrule In FrameMaker, an read/write rule that is part of another rule.

suffix Text that is automatically placed after the content of an element. In FrameMaker, a prefix is defined as part of the formatting of an element. *See also* prefix.

system identifier In markup, a way of identifying an external entity that's specific to the particular document or system. *See also* public identifier.

template In FrameMaker, a document used to create new documents. A template can include all the formats, structure descriptions, and other information you need to create a document.

<TEXT> In a FrameMaker element's general rule, indicates that the element can directly contain text characters and elements included by itself or its ancestors. `<TEXT>` corresponds to `#PCDATA` in a markup element's model group.

Text entity An entity whose replacement text can contain both data and markup.

text flow The text in a series of connected text frames. A text flow can also be contained in a single text frame, not connected to any other frame. A text flow with elements is a structured text flow.

text inset Text imported by reference.

<TEXTONLY> In a FrameMaker element's general rule, indicates that the element can directly contain text characters and cannot contain elements included by an ancestor. By default, on export `<TEXTONLY>` corresponds to a declared content of `RCDATA` in an SGML element's definition, or PCDATA in XML. On import FrameMaker translates declared content of `RCDATA` or `CDATA` to `<TEXTONLY>`.

valid document A structured document that conforms to all its content rules. Every element in the document must be valid. In FrameMaker, every structured flow must have a highest-level element that is allowed at the highest level.

valid element An element with contents that conform to its own content rules and to the inclusion and exclusion rules of all of its ancestors.

validating parser In markup, a software module that parses the markup of an XML or SGML document and determines that the document structure conforms to a provided DTD.

variable In FrameMaker, text that is defined once but can be used several times. Similar to some varieties of XML or SGML entity.

XML An acronym for Extensible Markup Language. By definition, XML is a subset of SGML.

XSLT An acronym for eXtensible Stylesheet Language: Transformations. It is a W3C language for transforming one XML document into another XML document. It can also transform an XML document into other text based formats including

MIF

.

Index

A

abstract types Schema mapping 233
all element Schema mapping 226
ampersand (&)
  in conversion tables 179
anchored frame (rule) 43
any element Schema mapping 230
anyType Schema mapping 222
application definition files ??–30
  contents of 9
  default information 11
  defining applications in 9
  document elements 13
  DTDs for import and export 14
  entity catalogs 15–16
  external entities 17–18
  filename extensions, specifying 21
  files for rules documents 22
  individual entities 16
  length of log files 30, 31, 32
  namespaces, enabling 22
  public identifiers 18
  read/write rules documents 22
  search path for external entities 19–20
  SGML declarations 24
  structure API clients 27
  templates for import 27
application files
  managing CSS 24
  Schema, specifying 23
  XSL transformation, specifying 25
asterisk (*)
  in conversion tables 179
attribute (rule) 46
attributes
  defaults in Schema 230
  for identifying overrides 184
  in conversion tables 181
  mapping of Schema to DTD 227
attributes, read/write rules for 34
  attribute 46
  drop 53
  fm attribute 76
  fm element 77
  implied value is 97
  is fm attribute 104
  is fm property 116
  is fm property value 126
  is fm value 139
  value 165

B

books, read/write rules for 35
  generate book 93
  output book processing instructions 147
  put element 93
  use processing instructions 93

C

CALS table model 235–240
  attribute structure 239
  colspec elements 238, 239
  element and attribute declarations 236
  element structure 238
  spanspec elements 238, 239
CALS tables
  read/write rules for 241–243
character formats
  wrapping text formatted without 184
character map (rule) 49
character set mapping 257–263
characters allowed
  in conversion tables 176
choice element Schema mapping 225
comma (,)
  in conversion tables 179
complex type Schema mapping 224, 231
  named 226
conversion tables 171–187
  adding rules to 175–182
  attributes in 181
  building tables from format tags with 186
  columns and rows in 171, 175
  documents for holding 172
  flagging format overrides with 184
  format and element tags in 173, 175, 177
  generating initial 173
  nesting graphics or tables with 185
  object type identifiers in 177
  order of rules in 172
  promoting graphics or tables with 183

qualifiers for element tags in 175, 181
root element 176
setting up from scratch 174
testing and correcting 186
updating 174
wrapping elements with 178
wrapping objects with 177
wrapping sequences with 179
wrapping untagged text with 184
cross-references, read/write rules for 35
fm element unwrap 77
fm property 80
is fm cross-reference element 109
is fm property 116
is fm property value 126
is fm value 139
value is 80
CSS
managing generation 24
CSS files 29
CSS import 25

D

default
SGML declaration 245–247
defaults
mapping of Schema to DTD 230
DOCTYPE elements 13
document type declarations (DTDs)
specifying location of 14
drop (rule) 53
drop content (rule) 55
DTD 222

E

element (rule) 56
element tags
in conversion tables 173, 175, 177
elements
defaults in Schema 230
mapping of Schema to DTD 228
elements, read/write rules for all 33
attribute 46
drop 53
drop content 55
element 56
fm element 77
is fm element 110
preserve fm element definition 148, 149
unwrap 162
encoding 29
of CSS files 29
end vertical straddle (rule) 59
entities
external files for 17–18
ISO public 249–256
searching for external files 19
searching for filename patterns 17
specifying location of 16
specifying search path for 19–20
entities, read/write rules for 36
drop 53
entity 61
entity name is 63
external data entity reference 72
is fm char 107
is fm reference element 128
is fm variable 140
reformat as plain text 154
reformat using target document catalogs 155
retain source document formatting 156
entity (rule) 61
entity catalogs
format of entries in 16
searching for 16
specifying location of 15–16
uses for 15
entity name is (rule) 63
equation (rule) 65
equations
in conversion tables 178
equations, read/write rules for 36
entity name is 63
equation 65
export dpi 66
export to file 69
fm property 80
is fm equation element 111
is fm property 116
is fm property value 126
is fm value 139
notation is 145
specify size in 157
value 165
value is 80
export dpi (rule) 66
export to file (rule) 69
exporting XML
XSL transformation 26
external data entity reference (rule) 72
external dtd (rule) 73

F

facet (rule) 74
fm attribute (rule) 76
fm element (rule) 77
fm element unwrap (rule) 77
fm marker (rule) 79
fm property (rule) 80
fm variable (rule) 92
fm version (rule) 93
footnotes
    in conversion tables 178
footnotes, read/write rules for 37
    is fm footnote element 113
format overrides, flagging in conversion tables 184
format tags
    in conversion tables 173, 175, 177

G

generate book (rule) 93
graphics
    nesting in conversion tables 185
    promoting in conversion tables 183
graphics, read/write rules for 37
    anchored frame 43
    entity name is 63
    export dpi 66
    export to file 69
    facet 74
    fm property 80
    is fm graphic element 114
    is fm property 116
    is fm property value 126
    is fm value 139
    notation is 145
    specify size in 157
    value 165
    value is 80
group element Schema mapping 224

I

impact of stylesheet element 25
implied value is (rule) 97
import
    Schema mapping 231
importing XML
    XSL transformation 26
include
    Schema mapping 231
include dtd (rule) 98
include sgml declaration (rule) 100
initial conversion tables 173
insert table part element (rule) 101
is fm attribute (rule) 104
is fm char (rule) 107
is fm cross-reference element (rule) 109
is fm element (rule) 110
is fm equation element (rule) 111
is fm footnote element (rule) 113
is fm graphic element (rule) 114
is fm marker element (rule) 115
is fm property (rule) 116
is fm property value (rule) 126
is fm reference element (rule) 128
is fm rubi element (rule) 130
is fm rubi group element (rule) 131
is fm system variable element (rule) 132
is fm table element (rule) 133
is fm table part element (rule) 135
is fm value (rule) 139
is fm variable (rule) 140
is processing instruction (rule) 141
ISO Latin-1 character set 257–263
ISO public entities 249–256
    declarations and rules 254–256
    default character formats 253
    default variable definitions 253
    entity declaration files 251
    entity read/write rules files 251
    format of entity rules 252

K

key element Schema mapping 233

L

line break (rule) 142
log files
    limiting length of 30, 31, 32

M

mapping of Schema elements 222
marker text is (rule) 143
markers, read/write rules for 38
    drop 53
    external data entity reference 72
    fm marker 79
    fm property 80
    is fm marker element 115
    is fm property 116
    is fm property value 126

is fm value 139
is processing instruction 141
marker text is 143
processing instruction 151
value 165
value is 80
markup language documents, read/write rules for 39
external dtd 73
include dtd 98
write structured document instance only 167
markup languages, translation to and from
cross-references ??–21

N

named attribute group Schema mapping 227
named complex type Schema mapping 226
namespaces
and Schema 221
extra attributes from Schema mapping 222
notation is (rule) 145

O

object type identifiers, in conversion tables 177
output book processing instructions (rule) 147

P

paragraph formats
building table structure from 186
parentheses
in conversion tables 179
plus sign (+)
in conversion tables 179
PostProcessing element 26
PreProcessing element 26
preserve fm element definition (rule) 148, 149
processing instruction (rule) 151
processing instructions (PIs), read/write rules for 39
drop 53
fm marker 79
is processing instruction 141
output book processing instructions 147
processing instruction 151
use processing instructions 93
PROMOTE keyword 183
proportional width resolution is (rule) 152
public identifiers 18
put element (rule) 93

Q

qualifiers, in conversion tables 171, 175, 181
question mark (?)
in conversion tables 179
quotation marks ("), in attribute values 181

R

read/write rules
documents for 22
for CALS tables 241–243
including files with 22
summary of 33–41
reader (rule) 153
redefine Schema mapping 231
reformat as plain text (rule) 154
reformat using target document catalogs (rule) 155
retain source document formatting (rule) 156
root element 176
Rubi groups, read/write rules for
is fm rubi element 130
is fm rubi group element 131

S

Schema
and namespaces 221
extra namespace attributes 222
mapping to DTD 222
mixed content models 229
specifying file location 221
structure application element 23
types not mapped 233
sequence element Schema mapping 224
SGML
defining an application 9
optional unsupported features 248
SGML declarations
default for FrameMaker 245–247
specifying location of 24
SGML documents, read/write rules for
include sgml declaration 100
SGML parser
concrete syntax variants 247
simple type Schema mapping 222
specify size in (rule) 157
start new row (rule) 159
start vertical straddle (rule) 160
structure API clients
specifying location of 27
structure applications
defining 9

structure, adding to documents. *See* conversion tables
stylesheet element
    impact on CSS import feature 25
stylesheets
    XSL 25
system variables
    in conversion tables 178

T

table ruling style is (rule) 161
tables
    building structure from format tags 186
    CALS attribute usage 235
    nesting in conversion tables 185
    promoting in conversion tables 183
tables, read/write rules for 40
    end vertical straddle 59
    fm property 80
    insert table part element 101
    is fm property 116
    is fm property value 126
    is fm table element 133
    is fm table part element 135
    is fm value 139
    proportional width resolution is 152
    start new row 159
    start vertical straddle 160
    table ruling style is 161
    use proportional widths 164
    value 165
    value is 80
templates
    specifying location of 27
text insets, read/write rules for 41
    entity 61
    reformat as plain text 154
    reformat using target document catalogs 155
    retain source document formatting 156
text, read/write rules for
    character map 49
    entity 61
    is fm char 107
    line break 142

U

unique element Schema mapping 233
untagged formatted text, wrapping 184
unwrap (rule) 162
use processing instructions (rule) 93
use proportional widths (rule) 164
user variables
    in conversion tables 178

V

value (rule) 165
value is (rule) 80
variables, read/write rules for 41
    drop 53
    entity 61
    fm element unwrap 77
    fm variable 92
    is fm system variable element 132
    is fm variable 140
vertical bar (|)
    in conversion tables 179

W

wildcard characters
    in conversion tables 176
wrapping with conversion tables
    document objects 177
    elements 178
    sequences of elements or paragraphs 179
    untagged formatted text 184
write structured document instance only (rule) 167
writer (rule) 168

X

XML
    defining an application 9
    specifying Schema location 221
    using CSS stylesheets 24
    XSL transformations 25
XML Schema, *See* Schema
XSL files
    associating with XML applications 25
XSL transformation (XSLT) 25

Legal notices

For legal notices, visit the Legal Notices page.