[MS-XAML-2009]:
XAML Object Mapping Specification 2009

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names.** The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Table of Contents

1 Preface

About This Specification

Xaml, the eXtensible Application Markup Language, is a system for representing structured information. This specification defines three aspects of Xaml:

- The Xaml Schema Information Set - a model for defining a particular Xaml vocabulary.
- The Xaml Information Set - a model for describing the information in a Xaml instance.
- The process for converting an XML [XML] document into the corresponding Xaml Information Set, as directed by one or more Xaml Schema Information Sets.

This specification does not define any particular Xaml vocabulary.

2 Language Notes

In this specification, the words that are used to define the significance of each particular requirement are capitalized. These words are used in accordance with their definitions in [RFC2119] and their meaning is reproduced here for convenience:

- MUST. This word, or the adjective "REQUIRED," means that the item is an absolute requirement of the specification.
- SHOULD. This word, or the adjective "RECOMMENDED," means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
- MAY. This word, or the adjective "OPTIONAL," means that this item is truly optional. For example, one implementation may choose to include the item because a particular marketplace or scenario requires it or because it enhances the product. Another implementation may omit the same item.

3 Overview

Xaml, the eXtensible Application Markup Language, is a system for representing structured information. This specification defines two abstract information models: the *Xaml Schema Information Set* model, and the *Xaml Information Set* model. The Xaml Information Set ('Xaml Infoset' for short) defines the structure of information that a Xaml instance can represent. The Xaml Schema Information Set allows specific Xaml vocabularies to be defined. This specification also defines a set of rules for transforming an XML document into a Xaml Information Set.

XML is a common format for Xaml. (The term "Xaml Document" refers to an XML document that represents a Xaml Information Set.) But while this specification does not define any other representations, any physical representation may be used as long as it can represent the information in the Xaml Information Set.

This first section of the specification describes the roles of the information sets, how they relate to applications that use Xaml, and how the transformation rules come into play.

3.1 Xaml Vocabularies and the Xaml Schema Information Set

This specification does not mandate any particular application or interpretation of a Xaml Document. Each individual application of Xaml will define its own *Xaml Vocabulary*. For example, Xaml could be used to define the structure of a graphical user interface, or it could be used to represent a list of pieces of music. This specification does not define any such application-specific vocabularies. This specification provides the information to enable such vocabularies to be defined.

A particular Xaml vocabulary is defined as a *Xaml Schema*. A schema defines the object types that may be used in a Xaml instance, determining the members and content each type supports. (Individual applications will likely go further, assigning meanings to element types. However, that is beyond the scope of a Xaml Schema.) Furthermore, some schema features are advisory - a schema may contain information which is not strictly required to process a Xaml Document, but which may be useful to tools. (For example, the information may enable compilers to provide better warnings, or for editors to offer better discoverability.)

A Xaml Schema is always associated with a particular namespace URI. XML representations of Xaml indicate their vocabulary through XML namespaces - the namespace URI of an element or attribute indicates the Xaml Schema to which that node belongs.

This specification does not define a Xaml Schema file format. Instead, this specification defines an abstract data model for schemas, the Xaml Schema Information Set. This specifies the information required to form a complete schema. It is defined in section 5, "Xaml Schema Information Set".

3.2 Xaml Instances, Xaml Documents and the Xaml Information Set

A *Xaml Instance* is a structured set of information, made up of the elements described in "Xaml Information Set", (section 6). The term does not mandate any particular representation. The term *Xaml Document* means an XML document that represents a *Xaml Instance*. The process for converting the XML in a Xaml Document into the Xaml Instance it represents is described in "Creating a Xaml Information Set from XML", (section 8).

The conversion process is defined in terms of the XML Information Set ([XML Infoset]), so the input XML document does not need to be a text stream formatted as required by the XML specification. Any representation may be used, as long as it can be mapped to the logical XML Information Set structure. For example, a Xaml Document could use a binary format, or it might be held in memory as a set of objects or data structures. As with Xaml Schemas, this specification is not concerned with the physical representation of a Xaml Document, only the logical structure.

3.3 Well-Formed and Valid Xaml Information Sets

This specification defines two characteristics that a Xaml Information Set may possess. It may be a *well-formed* Xaml Information Set, and it may be a *valid* Xaml Information Set.

A Xaml Information Set is well-formed if it conforms to all of the rules for well-formedness in "Xaml Information Set", (section 6).

A Xaml Information Set is valid with respect to one or more schemas if it conforms to all of the rules for validity in "Xaml Information Set", (section 6).

A Xaml Document, i.e., an XML document representing a Xaml Instance, is valid with respect to a set of schemas if it can be successfully converted into a Xaml Infoset using the process defined in 8, and the resulting Xaml Infoset is valid with respect to those schemas.

3.4 The Structure of Information in Xaml

There are obvious parallels between the world of XML and Xaml. Each have schemas, documents, and concepts of being well-formed and valid. The main difference is that the structure of the Xaml Information Set is more specialized than the XML Information Set. In particular, although there is no requirement that a Xaml instance be represented as objects in an object-oriented programming system, the Xaml Information Set has been designed to make such a mapping straightforward. Xaml Schemas are correspondingly specialized.

The structure of the information in a Xaml Information Set is described formally in section "Xaml Schema Information Set", (section 5), but as an aid to understanding, there follows a short, informal, non-normative description of the structure.

Information in a Xaml Instance is in one of three forms: objects, members, or text. The root of a Xaml Instance is an object. Objects can have members; an object's members are unordered. Members have values, which consist either of objects or text. Some members may have multiple values (forming an ordered sequence); in such cases, a single member's values may comprise a mixture of objects and text. Figure 1 shows an example XML representation of Xaml, and an illustration of the structure of the Xaml Infoset the XML represents.

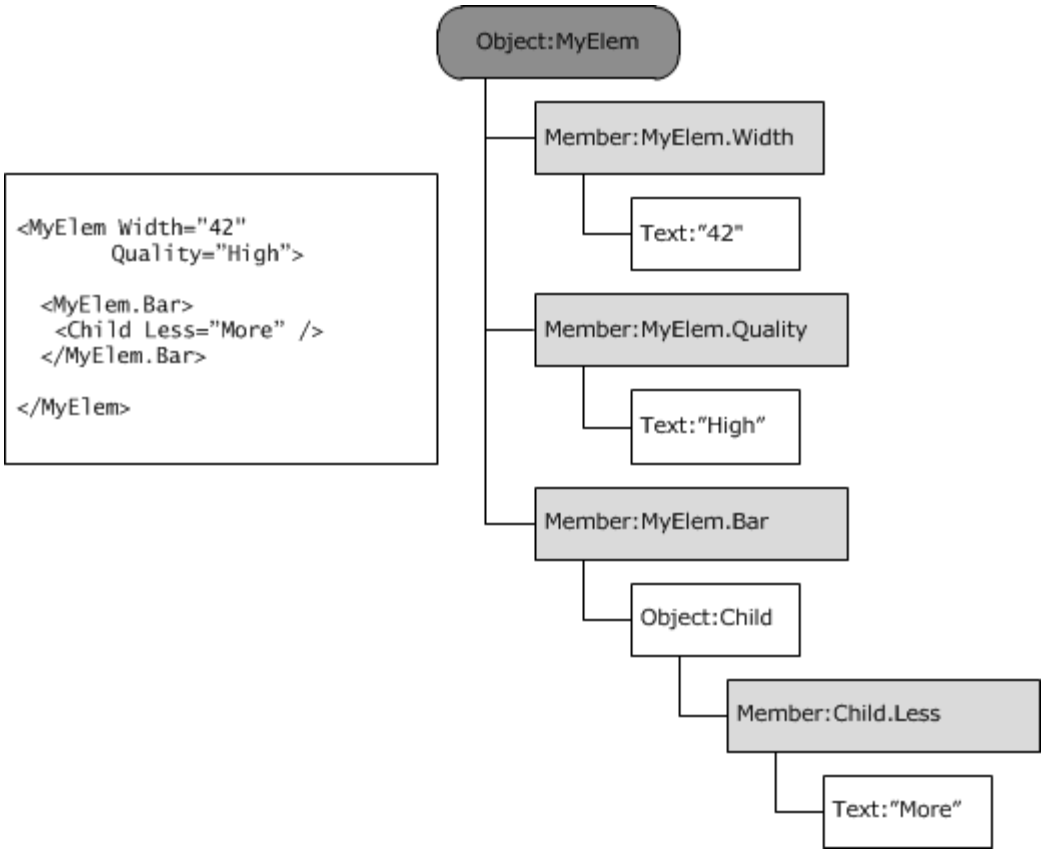

Figure 1: Structure of Xaml Information (non-normative)

Some non-normative sections of this specification use terminology common in object-oriented programming systems. An 'instance' of a type means an object of that type. 'Construction' refers to the act of creating an object.

3.5 A Note on Notation

When referring to Information Set properties (whether Xaml or XML), this specification uses bracketed notation. For example if 'elem' refers to an XML element in an XML information set, elem[attributes] refers to the [attributes] property of that element.

4 Information Set Type System

The Xaml Schema Information Set and the Xaml Information Set each define a data model. The models define various Information Item types. For example, the Xaml Schema Information Set defines a Schema Information Item (section 5.1).

Information Items have various properties, each of which has a type. In some cases, the type is another Information Item defined elsewhere in the data model. For example, the XamlMember Information Item (section 5.3) in the Xaml Schema Information Set has a [value type] property, and its type is another part of the model: a XamlType Information Item (section 5.2). However, these data models are not defined entirely in terms of themselves. For example, a XamlType Information Item has a [name] property whose type is XamlName. The following sections define the types such as these that are used in the information sets in this specification.

4.1 Text String

An ordered sequence of Unicode [Unicode] characters.

4.2 XamlName

A XamlName is a Text String that conforms to the following grammar (using the ABNF syntax defined in [RFC4234]):

```
      XamlName = NameStartChar *(NameChar)
NameStartChar = UnicodeLu / UnicodeLl / UnicodeLo
                / UnicodeLt / UnicodeNl / "_"
      NameChar = NameStartChar / UnicodeNd / UnicodeMn
                 / UnicodeMc / UnicodeLm
```

This assumes the following general category values as defined in the Unicode Character Database [UNICODE5.0.0/2007]:

Table 1: Unicode Category Abbreviations

Abbreviation	Description
UnicodeLu	Letter, Uppercase
UnicodeLl	Letter, Lowercase
UnicodeLt	Letter, Titlecase
UnicodeLm	Letter, Modifier
UnicodeLo	Letter, Other
UnicodeMn	Mark, Non-Spacing
UnicodeMc	Mark, Spacing Combining
UnicodeNd	Number, Decimal
UnicodeNl	Number, Letter

4.3 Namespace Uri

A Namespace Uri is a sequence of characters identifying an XML namespace. Implementations that process Xaml MUST accept any "namespace name" (as defined in section 2 of [XML Namespaces]) as a Namespace Uri. Implementations MAY tolerate as identifiers sequences of Unicode characters which are not valid namespace names.

4.4 Boolean

A truth value: either True or False.

4.5 Allowed Location

A value of this type represents an allowed location for a member in an XML representation--some members (e.g., the xml:lang Directive defined in section 7.3.13) may only be specified as attributes in XML, and not as member elements; some (e.g., the x:Items member defined in section 7.3.1) may not appear in XML at all, and are only used in a Xaml Information Set. Values of this type are one of: Any, AttributeOnly, InitialMemberElementsOnly, AttributeOrInitialMemberElementsOnly, or None.

IntitialMemberElementsOnly means that the member can be one of the initial member elements inside an element. As soon as content or a member element occurs that isn't an member with [allowed location] of InitialMemberElementsOnly or AttributeOrInitialMemberElementsOnly, no more (InitialMemberElementsOnly or AttributeOrInitialMemberElementsOnly) members may occur inside the element.

4.6 XML Namespace Mapping

A value of this type represents an XML namespace prefix and the corresponding XML namespace URI. It has two properties: [prefix] of type Text String, and [uri] of type Namespace Uri.

4.7 Set

Some properties have a type defined as a 'Set of' some type, or a list of types. For example, XamlType Information Item (section 5.2)has a [types assignable to] property whose type is 'Set of XamlType Information Items'. Such a property may contain any number of values of the type or types in question. Order is not significant in a set. Sets do not contain duplicates--any given value is either in a set or it is not. Sets may be empty.

4.8 Ordered Collection

Some properties have a type defined as an 'Ordered Collection of' some type, or a list of types. For example, Member Node Information Item (section 6.3) has a [values] property, whose type is defined as 'Ordered collection of information items; each item may be either an Object Node Information Item, or a Text Node Information Item'. Such properties may contain any number of values of the type or types in question. The values are strictly ordered. Ordered collections may contain duplicates, for example, the second and fourth item in an ordered collection may be the same value. An ordered collection may be empty.

5 Xaml Schema Information Set

A Xaml Schema is an abstract definition of a set of Xaml Instances. A Xaml Infoset is said to be an instance of a Xaml Schema if it conforms to all of the rules for well-formedness and validity in section (6), "Xaml Information Set".

This specification does not mandate a representation or format for a Xaml Schema. Instead, it defines an abstract data model called the Xaml Schema Information Set, or 'Schema Infoset' for short, which defines the elements that make up a Xaml Schema.

A Xaml Schema Information Set can contain five kinds of items. These are listed in Table 2.

Table 2: Xaml Schema Information Set Item Kinds

Item Kind	Purpose (non-normative)
Schema Information Item	Identifies a schema, and defines which information items constitute that schema's definition.
XamlType Information Item	Describes a type of element that instances of this schema may contain.
XamlMember Information Item	Describes a member that may be applied to elements in instances of this schema.
Text Syntax Information Item	Describes the valid textual representations of the values of a particular member or type.
Constructor Information Item	Describes the available constructor forms for a type.

The information items that make up the Xaml Schema Information Set have identity. For example, given two XamlMember Information Items, it is meaningful to ask if their [value type] properties each refer to the same XamlType Information Item (or more informally, whether the two members have the same type). Since this specification does not mandate any particular representation for schema infosets, implementations are free to represent this in any way. For example, a programming system might choose to represent item identity through object identity where such a concept is supported; a serialized representation might choose to add identifiers that do not correspond directly to infoset properties purely to handle item identity.

The following sections describe the data properties that make up each of the information item types. These sections also define constraints a Xaml Schema must meet.

In some cases, notes are provided to describe the purpose of certain data properties in more detail. This is done in situations where the normative interpretation of these properties is defined in other sections, but where the intended meaning of the properties would be hard to infer. These notes are provided purely as an aid to understanding, and are marked as 'non-normative' to indicate that they does not constitute a formal part of the Xaml Schema Infoset specification. The 'Purpose' column of each table defining an information item has a similar role, and is also non-normative.

5.1 Schema Information Item

Each Xaml Schema MUST have one Schema Information Item. The Schema Information Item is the root of a schema's definition, defining which other items belong to this schema.

Table 3: Schema Information Item Properties

Name	Type	Purpose (non-normative)
[target namespace]	Namespace Uri	The target XML namespace for this schema.
[types]	Set of XamlType Information Items	The object types that instances of this schema may contain.
[assignable types]	Set of XamlType Information Items	Types that are not used directly as values in instances of this schema, but which are used by XamlMember Information Items for assignability purposes.
[directives]	Set of XamlMember Information Items	The directive attributes that may be applied to objects in instances of this schema.
[compatible with schemas]	Set of Schema Information Items	The schemas with which this schema is considered compatible.

5.1.1 Constraints

A Xaml Schema Information Set'sSchema Information Item MUST conform to the rules defined in this section.

5.1.1.1 Information Set Properties Must Be of Correct Type

Each property of a Schema Information Item MUST have a value of the type specified for that property in Table 3.

5.1.1.2 Members in [directives] Must Be Directives

For each XamlMember Information Item 'd' in [directives], d[is directive] MUST be True.

5.1.1.3 Type Names Must Be Unique

For each XamlType Information Item 't' in [types] the value of t[name] MUST be different from the [name] of any other XamlType Information Item in [types].

5.1.1.4 Directive Names Must Be Unique

For each XamlMember Information Item 'd' in [directives] the value of d[name] MUST be different from the [name] of any other XamlMember Information Item in [directives].

5.1.2 Notes (Non-Normative)

The [compatible with schemas] property addresses two scenarios. It is used when new versions of a vocabulary are developed that are compatible with older versions. It is also used to handle cases where multiple distinct namespaces may identify structurally identical schemas.

The types in [types] can be used for assignability purposes as well as the types in [assignable types]. The significance of [assignable types] is that types in this property do not support the full range of features that types in [types] do.

5.2 XamlType Information Item

A XamlType Information Item defines a data type. For example, an Object Node Information Item (section 6.2) has a [type] property that refers to a XamlType Information Item.

Table 4: XamlType Information Item Properties

Name	Type	Purpose (non-normative)
[name]	XamlName	The name that represents this type. Case is significant.
[types assignable to]	Set of XamlType Information Items	The types to which instances of this type are considered assignable.
[is default constructible]	Boolean	When True, this type may always be used as the type for an object node. When False, constraints apply, which are described later in this specification.
[is nullable]	Boolean	True if members of this type may have a Null value.
[text syntax]	Null, or Text Syntax Information Item	The text syntax that defines how instances of this type can be represented as text.
[members]	Set of XamlMember Information Items	The members available on this type.
[content property]	Null, or a XamlMember Information Item	The member to which content of an element of this type can be assigned. (Allows XML representations to omit the member element.)
[dictionary key property]	Null, or a XamlMember Information Item	The member that acts as the key if an element of this type is added to a dictionary without a key being specified explicitly. The designated member is effectively an alias for the x:Key Directive (section 7.3.5).
[name property]	Null, or a XamlMember Information Item	The member that, if set, holds the name of an element of this type. The designated member is effectively an alias for the x:Name Directive (section 7.3.4).
[xml lang property]	Null, or a XamlMember Information Item	The member that holds the value of the xml:lang attribute (when present) – the designated member is effectively an alias for the xml:lang Directive (section 7.3.13).
[trim surrounding whitespace]	Boolean	True if whitespace immediately before and after elements of this type in an XML representation should be removed.
[is whitespace significant collection]	Boolean	True if, when a Xaml processor reads an XML representation of an element of this type, whitespace content should not be collapsed.
[is list]	Boolean	True if elements of this type contain an ordered sequence of items.
[is dictionary]	Boolean	True if elements of this type contain a set of items, each identified by a key.
[allowed types]	Set of XamlType	The types that can be added as items inside a list or

Name	Type	Purpose (non-normative)
	Information Items	dictionary. (Only used if [is list] or [is dictionary] is True.)
[**allowed key types**]	Set of XamlType Information Items	The types that can be used as keys. (Only used if [is dictionary] is True.)
[is xdata]	Boolean	True if elements of this type contain literal XML data.
[is name scope]	Boolean	Used to determine the scope in which values of the x:Name Directive (section 7.3.4) must be unique.
[constructors]	Set of Constructor Information Items	The constructors that can be used to create instances of this type.
[return value type]	Null, or a XamlType Information Item	The type of value provided by this type. (Only used for markup extensions – see the x:MarkupExtension (section 7.2.22) type.)
[is generic]	Boolean	True if elements of this type can accept x:TypeArguments Directive (section 7.3.11).

The [types assignable to] property contains the complete set of types from this schema to which instances of this type are assignable. If a vocabulary wishes to provide common object-oriented semantics, such as having a type be assignable to all the types to which its base class is assignable, it must make that explicit. However, there is an additional complexity regarding schemas that list other schemas in their [compatible with schemas]. Types do not include types from the schemas with which they are compatible with their [types assignable to]. Instead, this specification presumes that equivalently named types in compatible schemas are compatible. To simplify the validity checks that presume this, the following functions are defined:

$$
\begin{aligned}
&\mathit{isCompatibleWith}(t1, t2) \equiv \\
&\quad (t1[\text{name}] = t2[\text{name}]) \wedge \\
&\quad (\forall s1 \in [s : (s \textit{ is a } \text{Schema Information Item}) \wedge s[\text{types}] \in t1] \\
&\quad (\forall s2 \in [s : (s \textit{ is a } \text{Schema Information Item}) \wedge s[\text{types}] \in t2] \\
&\qquad ((s1 = s2) \\
&\qquad\quad \vee \\
&\qquad (s1 \in s2[\text{compatible with schemas}]) \\
&\qquad\quad \vee \\
&\qquad (s2 \in s1[\text{compatible with schemas}]))))
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{isAssignableTo}(tFrom, tTo) \equiv \mathit{isCompatibleWith}(tFrom, tTo) \vee \\
&\qquad (\exists\, cTo \mid (cTo \in tFrom[\text{types assignable to}]) \wedge \mathit{isCompatibleWith}(cTo, tTo))
\end{aligned}
$$

Informally, a type tFrom is assignable to a type tTo if either tFrom and tTo are compatible, or if tFrom [types assignable to] contains a type which is compatible with tTo; types are compatible with types that have the same name and compatible schemas.

5.2.1 Constraints

The XamlType Information Items in a Xaml Schema Information Set MUST conform to the rules defined in this section.

5.2.1.1 Information Set Properties Must Be of Correct Type

Each property of a XamlType Information Item MUST have a value of the type specified for that property in Table 4.

5.2.1.2 Content Member Must Be Available

If a type's [content property] is not Null, that type's [members] MUST contain the member in [content property].

5.2.1.3 Name Member Must Be Available

If a type's [name property] is not Null, that type's [members] MUST contain the member in [name property].

5.2.1.4 Content Member Mutually Exclusive with List and Dictionary

If [content property] is not Null, [is list] and [is dictionary] MUST both be False.

5.2.1.5 List and Dictionary Mutually Exclusive

If [is list] is True, [is dictionary] MUST be False (from which the converse follows: if [is dictionary] is True, [is list] MUST be False.)

5.2.1.6 Allowed Types Only Used on Lists and Dictionaries

[allowed types] MUST be empty unless either [is list] or [is dictionary] is True.

5.2.1.7 Allowed Key Types Only Used on Lists and Dictionaries

[allowed key types] MUST be empty unless [is dictionary] is True.

5.2.1.8 Return Value Type Required on Markup Extension

If [types assignable to] contains the intrinsic x:MarkupExtension (section 7.2.22) type, [return value type] MUST NOT be Null.

5.2.1.9 Return Value Type Only Used on Markup Extension

If [types assignable to] does not contain the intrinsic x:MarkupExtension (section 7.2.22) type, [return value type] MUST be Null.

5.2.1.10 Only Markup Extensions Can Have Constructors

If [types assignable to] does not contain the intrinsic x:MarkupExtension (section 7.2.22) type, [constructors] MUST be empty.

5.2.1.11 No Two Constructors May Have the Same Number of Arguments

For each Constructor Information Item ctor in [constructors], [constructors] MUST NOT contain any other Constructor Information Items that have the same number of items in their [arguments] as there are in ctor[arguments].

5.2.2 Notes (Non-Normative)

Both [members] and [types assignable to] are comprehensive within a schema, i.e., types do not automatically inherit everything that types in [types assignable to] have. For example, suppose type B has member BP, and a type D has a [types assignable to] containing B; if D wishes to make BP available, D[members] must contain BP. The reason for this is not to force Xaml vocabularies to use 'normal' inheritance rules. Vocabularies that wish to offer a typical object-oriented style of inheritance are free to do so; they must simply be explicit.

However, type assignability is honored in Xaml Instances. For example, consider [allowed types]. Items in a list or dictionary may also be of types that are assignable to types in [allowed types], as determined by each item type's [types assignable to] property. Likewise, for [allowed key types], key values may also be of types that are assignable to types in this list.

In short, Xaml Schemas are required to be explicit; validation of Xaml Instances does whatever the schema says.

5.3 XamlMember Information Item

A XamlMember Information Item provides information about a member. Members are either defined by a particular XamlType Information Item, or they are directives.

Table 5 XamlMember Information Item Properties

Name	Type	Purpose (non-normative)
[name]	XamlName	The name of the member.
[owner type]	Null or XamlType Information Item	The type that defines this member, or Null if [is directive] is True.
[value type]	XamlType Information Item	The type that values for this member must be assignable to.
[text syntax]	Null, or Text Syntax Information Item	A member-specific text syntax that defines how this member can be represented as text. (If present, this takes precedence over [value type][text syntax].)
[is read only]	Boolean	True if the member cannot be set. (Only used for lists and dictionaries.)
[is static]	Boolean	True if the member is associated directly with the defining type, and not with any particular element.
[is attachable]	Boolean	True if this member may be applied to types other than those compatible with the owner type.
[target type]	Null, or XamlType Information Item	The member may be attached to types compatible with this type. (Only used for attachable members.)
[allowed	Allowed Location	Indicates how the member may be represented in XML.

Name	Type	Purpose (non-normative)
location]		
[is event]	Boolean	True if this member is used to define a response to some event such as user input. (The interpretation of event members is determined by individual Xaml processors.)
[is directive]	Boolean	True if this is a directive.

5.3.1 Constraints

The XamlMember Information Items in a Xaml Schema Information Set MUST conform to the rules defined in this section.

5.3.1.1 Information Set Properties Must Be of Correct Type

Each property of a XamlMember Information Item MUST have a value of the type specified for that property in Table 5.

5.3.1.2 Member Names Must Be Unique

If [is directive] is False, the value of [name] MUST be different from the [name] of any other XamlMember Information Item in [owner type][members].

5.3.1.3 Member Kind

At most one of [is attachable], [is event], and [is directive] can be True.

5.3.1.4 Must Have Owner Type or Be Directive

If [is directive] is False, [owner type] MUST NOT be Null. If [is directive] is True, [owner type] MUST be Null.

5.3.1.5 Owner Type Must Own Member

If [owner type] is not Null, this XamlMember Information Item MUST be in [owner type][members].

5.3.1.6 Only List, Dictionary, or Static Members May Be Read-Only

If neither [value type][is list] nor [value type][is dictionary], nor [is static] is True, [is read only] MUST be False.

5.3.1.7 Properties Required by Attachable Members

If [is attachable] is True, [target type] MUST NOT be Null.

5.3.1.8 Properties Unique to Attachable Members

If [is attachable] is False, [target type] MUST be Null.

5.3.1.9 Event Type Must Be XamlEvent

If [is event] is True, [value type] MUST be the intrinsic x:XamlEvent (section 7.2.21).

5.3.1.10 Properties Not Supported by Directives

If [is directive] is True, [owner type] MUST be Null.

5.3.2 Notes (Non-Normative)

A member whose [value type] is a list or dictionary can be read-only, because it is still possible to modify the contents of the list or dictionary, even when the member that holds it cannot be changed. In this case, being read-only merely prevents us from supplying our own list or dictionary.

Members for which [is attachable] is True can be applied in a non-attached manner on instances of the type that defines the member, as long as the member target is compatible with the member's [target type]. E.g., if a type Canvas defines an attachable Canvas.Left member, that member can be used in a non-attached fashion on a Canvas.

Directives are similar to members - in the XML representation, they are set on object nodes using the same syntax as a member, and in the Xaml Information Set, directives are represented as member nodes. However, unlike normal members, a directive is not owned by any particular type. Directives are typically associated with special behaviors in the Xaml handling. For example, this specification defines some intrinsic directives in Intrinsic Schema Information Items, (section 7), such as x:Key (section 7.3.5), which determines how dictionary items are handled.

5.4 Text Syntax Information Item

A Text Syntax Information Item describes the way in which the values for a particular type or member can be represented in a Xaml Instance (section 3.2).

Table 6: Text Syntax Information Item Properties

Name	Type	Purpose (non-normative)
[values]	Set of Value Syntax Information Items	Fixed textual values that are known to be valid.
[patterns]	Set of Pattern Syntax Information Items	Pattern-based expression of valid values.

5.4.1 Constraints

The Text Syntax Information Items in a Xaml Schema Information Set MUST conform to the rules defined in this section.

5.4.1.1 Information Set Properties Must Be of Correct Type

Each property of a Text Syntax Information Item MUST have a value of the type specified for that property in Table 6.

5.4.2 Notes (Non-Normative)

Members and types can have Text Syntax Information Items associated with them. The [values] property contains a list of literal values the member may use. This could be used for a member that corresponded to an enumerated type in a programming system--[values] would contain one string for each enumeration member.

The [patterns] property is used when the set of valid values is either not closed, or would be too large to enumerate. For example, it would not be practical to describe the valid values for a numeric

field by putting every possible number representation into [values]. Instead, [patterns] would capture the valid values.

A Text Syntax Information Item may contain multiple [values] and [patterns]. A text value is considered a match if it matches at least one value or pattern. (All of this is formalized in section (6), Xaml Information Set.)

5.5 Value Syntax Information Item

A Value Syntax Information Item describes a possible value. All members of the [values] collection in a Text Syntax Information Item are Value Syntax Information Items.

Table 7: Value Syntax Information Item Properties

Name	Type	Purpose (non-normative)
[text]	Text String	Fixed textual value that is known to be valid.
[trim whitespace]	Boolean	True if whitespace before and after the value can be ignored.
[is case sensitive]	Boolean	True if the value must match exactly, false if the value may be lowercase or capitals.

5.5.1 Constraints

The Text Syntax Information Items in a Xaml Schema Information Set MUST conform to the rules defined in this section.

5.5.1.1 Information Set Properties Must Be of Correct Type

Each property of a Text Syntax Information Item MUST have a value of the type specified for that property in Table 7.

5.6 Pattern Syntax Information Item

Pattern Syntax Information Item

Table 8: Pattern Syntax Information Item Properties

Name	Type	Purpose (non-normative)
[pattern]	Text String	Pattern-based expression of valid values.
[trim whitespace]	Boolean	True if whitespace before and after the value can be ignored.
[is case sensitive]	Boolean	True if the value must match exactly, false if the value may be lowercase or capitals.

The string in [pattern] is a regular expression. Xaml Schema Infoset regular expressions use the same formulation as those in XML Schema Definitions. See Appendix F of Part 2 of the XML Schema specification [XML Schema Part 2].

5.6.1 Constraints

The Pattern Syntax Information Items (section 5.6) in an Xaml Schema Information Set (section 5) MUST conform to the rules defined in this section.

5.6.1.1 Information Set Properties Must Be of Correct Type

Each property of a Pattern Syntax Information Item (section 5.6) MUST have a value of the type specified for that property in Table 8.

5.7 Constructor Information Item

A Constructor Information Item defines a parameter list that can be used to construct a particular type.

Table 9: Constructor Information Item Properties

Name	Type	Purpose (non-normative)
[arguments]	Ordered Collection of XamlType Information Items	The types of the parameters for this constructor.

5.7.1 Constraints

The Constructor Information Items in a Xaml Schema Information Set MUST conform to the rules defined in this section.

5.7.1.1 Information Set Properties Must Be of Correct Type

Each property of a Constructor Information Item MUST have a value of the type specified for that property in Table 9.

6 Xaml Information Set

A Xaml Information Set represents the information contained in a Xaml Instance. A Xaml Information Set can contain four kinds of items. These are listed in Table 10.

Table 10: Xaml Information Set Item Kinds

Item Kind	Purpose (non-normative)
Document Information Item	Represents a Xaml Instance.
Object Node Information Item	Describes an object in the Xaml Instance.
Member Node Information Item	Describes a member that has been set on a particular object in the Xaml Instance.
Text Node Information Item	Describes textual or whitespace content in the Xaml Instance.

The following sections describe the data properties that make up each of the information item types. These sections also define constraints that all Xaml Instances are required to meet, and further constraints that must be met for a Xaml Instance to be considered valid.

In some cases, notes are provided to describe the purpose of certain data properties in more detail. This is done in situations where the normative interpretation of these properties is defined in other sections, but where the intended meaning of the properties would be hard to infer. These notes are provided purely as an aid to understanding, and are marked as 'non-normative' to indicate that they do not constitute a formal part of the Xaml infoset specification. The 'Purpose' column of each table defining an information item serves a similar role, and is also non-normative.

Note that many of the properties in the Xaml Information Set information items refer to information items from the Xaml Schema Information Set.

6.1 Document Information Item

Each Xaml Instance has exactly one Document Information Item. It represents the document, and provides access to the document's root.

Table 11: Document Information Item Properties

Name	Type	Purpose (non-normative)
[document object]	Object Node Information Item	The root object of the Xaml Instance.

6.1.1 Constraints

A well-formed Xaml Information Set MUST conform to the rules defined in this section.

6.1.1.1 Information Set Properties Must Be of Correct Type

Each property of a Document Information Item MUST have a value of the type specified for that property in Table 11.

6.1.1.2 Xaml Must Have Tree Structure

Consider a Xaml Information Set as a directed graph, where Object Node Information Items form nodes, and edges are described by the Member Node Information Items in each Object Node Information Item's [member nodes]. This graph MUST NOT contain cycles. Furthermore, any single Object Node Information Item MUST NOT be referred to by more than one Member Node Information Items.

6.2 Object Node Information Item

Object nodes are one of the three main forms of information in a Xaml Information Set (the others being member nodes and text nodes). Each object node is represented by an Object Node Information Item.

Table 12: Object Node Information Item Properties

Name	Type	Purpose (non-normative)
[type]	XamlType Information Item	Schema-defined type of this object.
[member nodes]	Set of Member Node Information Items	The members that have been set on this object.
[parent member]	Null, or Member Node Information Item	The member node for which this object is a value.
[is retrieved]	Boolean	True if this object node does not represent a new object to be created - instead, the members in [member nodes] are to be set on an existing object instance
[xml namespace mappings]	Set of XML namespace mappings	The XML namespace mappings in effect at this element.

Some of the constraints that follow refer to a 'root' node. This is the node in the Document Information Item's [document object] property. Although Object Node Information Items do not contain a reference back to their containing Document Information Item, the root node can still be located with the following process:

Let 'current node' be the node for which the root is to be found.

If the [parent member] of current node is Null, the current node is the root node; if not, continue to the next step.

Take the [parent member][parent object] of the current node, make that the new current node, and then repeat from step 2.

6.2.1 Constraints

The Object Node Information Items in a well-formed Xaml Information Set MUST conform to the rules defined in this section.

6.2.1.1 Information Set Properties Must Be of Correct Type

Each property of an Object Node Information Item MUST have a value of the type specified for that property in Table 12.

6.2.1.2 Events Not Allowed Unless Root Has x:Class

[member nodes] MUST NOT contain any Member Node Information Items (section 6.3) whose [is event] is True unless the root node's [member nodes] contains a Member Node Information Item whose [member] is the x:Class directiveXamlMember Information Item (section 5.3).

6.2.1.3 Cannot Have Multiple Member Nodes with Same Member

The Member Node Information Items in [member nodes] MUST all have different [member] items.

6.2.1.4 Parent Must Contain This Node

This Object Node Information Item MUST be in [parent member][values].

6.2.2 Validity Constraints

The Object Node Information Items in a valid Xaml Information Set MUST conform to the rules defined in this section.

6.2.2.1 Cannot Set Both x:Name and Name Member

If [member nodes] contains a Member Node Information Item whose [member] is the intrinsic x:Name Directive (section 7.3.4), [member nodes] MUST NOT also contain an item whose [member] is this node's [type][name property].

6.2.2.2 Cannot Set Both xml:lang and Language Member

If [member nodes] contains an item whose [member] is the intrinsic xml:lang Directive (section 7.3.13), [member nodes] MUST NOT also contain an item whose [member] is this node's [type][xml lang property].

6.2.2.3 Types without Default Constructor Require Constructor Parameters

If the node's [type][is default constructible] is False, the node's [member nodes] MUST contain an item whose [member] is the intrinsic x:PositionalParameters7.3.2.

6.2.2.4 Constructor Parameters Must Match Constructor Info

If the node's [member nodes] contain an item whose [member] is the intrinsic x:PositionalParameters (section 7.3.2), call that item positionalParameters. The node's [type][constructors] MUST contain a Constructor Information Item for which the number of XamlType Information Items in constructorInfo[arguments] is the same as the number of items in positionalParameters[values]. Call that item constructorInfo.

For each XamlType Information Item argType in constructorInfo[arguments], and for the item argValue at the same offset into the positionalParameters[values] sequence, the following apply:

- If argValue is a Text Node Information Item, one of the following MUST be true:
  - argType is the intrinsic x:String (section 7.2.7).

- argType[text syntax] is non-Null.

- Otherwise, argValue is an Object Node Information Item, and one of the following MUST be true:
  - isAssignableTo(argValue[type], argType) is True.
  - argValue[type][types assignable to] contains the x:MarkupExtension (section 7.2.22) intrinsic type, and isAssignableTo(argValue[type][return value type], argType) is True.

6.2.2.5 Initialization Text Must Match Text Syntax

If the node's [member nodes] contain an item whose [member] is the intrinsic x:Initialization (section 7.3.3), that item's [values] MUST contain a single Text Node Information Item. Let initializationText be the [text] of that text node.

Let textSyntax be a Text Syntax Information Item defined as follows:

- If [parent member][member][text syntax] is not Null, then that is the value of textSyntax.
- Otherwise, [type][text syntax] MUST NOT be Null, and that is the value of textSyntax.

The text value in initializationText MUST either match one of the entries in textSyntax[values], or match one of the entries in textSyntax[patterns]. The initializationText value is determined to be a match for a Value Syntax Information Item 'valueSyntax' from textSyntax[values] with the following process:

- Let trimmedInput be a string defined as follows:
  - If valueSyntax[trim whitespace] is true, trimmedInput is formed by removing any whitespace from the start and end of initializationText.
  - Otherwise, trimmedInput is initializationText.
- Let casedSyntaxValue and casedInput be two strings defined as follows:
  - If valueSyntax[is case sensitive] is true, casedSyntaxValue is valueSyntax[text], and casedInput is trimmedInput.
  - Otherwise, caseSyntaxValue is formed by converting any letters in valueSyntax[text] to their uppercase equivalents, and caseInput is formed by converting any letters in trimmedInput to their uppercase equivalents.
- initializationText matches valueSyntax if and only if casedSyntaxValue is equal to casedInput.

The initializationText value is determined to be a match for a Pattern Syntax Information Item 'patternSyntax' from textSyntax[patterns] with the following process:

- Let trimmedInput be a string defined as follows:
  - If patternSyntax[trim whitespace] is true, trimmedInput is formed by removing any whitespace from the start and end of initializationText.
  - Otherwise, trimmedInput is initializationText.
- Let casedInput be a string defined as follows:
  - If patternSyntax[is case sensitive] is true, casedInput is trimmedInput.

- Otherwise, casedInput is formed by converting any letters in trimmedInput to their uppercase equivalents.

- initializationText matches patternSyntax if and only if casedInput is a match for the regular expression in patternSyntax[pattern]; Xaml Schema Infoset regular expressions use the same formulation as those in XML Schema Definitions. See Appendix F of Part 2 of the XML Schema specification [XML Schema Part 2].

6.2.2.6 Cannot Provide Initialization Text and Other Member Values

If the node's [member nodes] contain an item whose [member] is the intrinsic x:Initialization (section 7.3.3), [member nodes] MUST NOT contain any other items other than the following three optional items: an item whose [member] is the intrinsic x:Key Directive (section 7.3.5); an item whose [member] is the intrinsic x:Uid Directive (section 7.3.6); and an item whose [member] [is attachable] is true.

6.2.2.7 x:XData Only Valid in XData Members

If the node's [type] is the intrinsic x:XData (section 7.2.24), [parent member][value type][is xdata] MUST be True.

6.2.2.8 x:TypeExtension Must Have Valid Type

If the node's [type] is the intrinsic x:TypeExtension (section 7.2.3), [member nodes] MUST contain exactly one Member Node Information Item whose [member] is one of the following:

- The intrinsic x:PositionalParameters (section 7.3.2).
- The intrinsic TypeExtension.Type (section 7.3.21).
- The intrinsic TypeExtension.TypeName (section 7.3.22).

[member nodes] MUST NOT contain more than one of the above.

6.2.2.9 x:StaticExtension Must Have Valid Member

If the node's [type] is the intrinsic x:StaticExtension (section 7.2.2), then there MUST exist a Text Node Information Item memberTextNode defined as follows:

- If the node's [member nodes] contains a single Member Node Information Item whose [member] is the intrinsic x:PositionalParameters (section 7.3.2), and this Member Node Information Item contains a single Text Node Information Item, let memberTextNode be that Text Node Information Item.
- Otherwise, the node's [member nodes] MUST include a Member Node Information Item whose [member] is the intrinsic StaticExtension.Member (section 7.3.20), and this Member Node Information Item MUST contain a single Text Node Information Item; let memberTextNode be that Text Node Information Item.

Let memberText be memberTextNode[text]. MemberText MUST be a DottedXamlName (section 8.5.1), and the DottedXamlName's typename part MUST be a valid QName (as defined in section 4 of [XML Namespaces]).

Let memberSchema be defined as follows:

- If the typename part is a PrefixedName (as defined in section 4 of [XML Namespaces]), then:

- This node's [xml namespace mappings] MUST contain an XML namespace mapping whose [prefix] matches the typename's Prefix (as defined in section 4 of [XML Namespaces]); let nsUri be the [uri] of that XML namespace mapping.
- A schema MUST be available whose [target namespace] matches nsUri. Let memberSchema be that schema.

- Otherwise, [xml namespace mappings] MUST contain a mapping whose [prefix] is the empty string, and a schema MUST be available whose [target namespace] matches the [namespace name] of that mapping; let memberSchema be that schema.

memberSchema[types] MUST contain a XamlType Information Item (section 5.2) whose [name] matches the DottedXamlName's typename part's LocalName (as defined in section 4 of [XML Namespaces]) in memberText. Call that XamlType Information Item memberType.

memberType[members] MUST contain a XamlMember Information Item whose [name] matches the DottedXamlName's membername part in MemberText. Call that XamlMember Information Item (section 5.2) memberInfo.

isAssignableTo (memberInfo[value type], [parent member][member][value type]) MUST be True.

6.2.2.10 Array Contents Must Be of Correct Type

If an Object Node Information Item's [type] is the intrinsic x:ArrayExtension (section 7.2.1), and if its [member nodes] contains a member 'arrayItems' whose [member] is the intrinsic ArrayExtension.Items (section 7.3.18), and if its [member nodes] also contains a member 'arrayType' whose [member] is the intrinsic ArrayExtension.Type (section 7.3.19), then for each Object Node Information Item 'item' in 'arrayItems' [values] isAssignableTo(item[type], arrayType) MUST be True.

6.2.2.11 Only Retrieved Objects May Use Assignable Types

If the node's [type] is a member of one or more schemas' [assignable types] and is not a member of any schema's [types], the node's [is retrieved] MUST be True.

6.2.3 Note (non-normative)

Since Xaml does not need to be represented as XML, it may seem odd to have an [xml namespace mappings] property. The reason for this is to enable Xaml to use XML namespace prefixes as a shorthand for referring to schemas.

6.3 Member Node Information Item

A Member Node Information Item represents one of a number of features. Properties, directives, events, or initialization strings are all represented as Member Node Information Items.

Table 13: Member Node Information Item Properties

Name	Type	Purpose (non-normative)
[member]	XamlMember Information Item	Xaml Schema Infoset information item that defines this member.
[parent object]	Object Node Information Item	The object on which this member has been set.

Name	Type	Purpose (non-normative)
[values]	Ordered Collection of information items; each item may be either an Object Node Information Item, or a Text Node Information Item	The value or values being set by this member node.
[xml namespace mappings]	Set of XML namespace mappings	The XML namespace mappings in effect at this member.

6.3.1 Constraints

The Member Node Information Items in a well-formed Xaml Information Set MUST conform to the rules defined in this section.

6.3.1.1 Information Set Properties Must Be of Correct Type

Each property of a Member Node Information Item MUST have a value of the type specified for that property in Table 13.

6.3.1.2 Multiple Values Only Allowed in List Content, Dictionary Content, or Constructor Arguments

If [member] is not one of the x:Items (section 7.3.1), x:DirectiveChildren (section 7.3.12), or x:PositionalParameters (section 7.3.2) intrinsic items, then [values] MUST contain exactly one item. (Otherwise, it can contain 0 or more such items.)

Note All members may contain any number of objects of intrinsic type x:Code (section 7.2.23).

6.3.1.3 Intrinsic x:Items Member Only Allowed in List or Dictionary

If [member] is the intrinsic x:Items (section 7.3.1) member type, then either [parent object][type][is dictionary] or [parent object][type][is list] MUST be True.

6.3.1.4 Dictionary Content Rules

If ([parent object][type][is dictionary] and [member] is the instrinsic x:Items (section 7.3.1)), the following rules apply:

- [values] MUST NOT contain any Text Node items.
- Each Object Node Information Item 'dictItem' in [values] MUST match at least one of the following (and let the first of these rules that matches define keyMemberNode for that dictItem):
  - dictItem[member nodes] contains a Member Node Information Item that is the x:Key directive (defined in section 7.3.5).
  - dictItem[member nodes] contains a Member Node Information Item whose [member] is dictItem[type][dictionary key property].
- For each Object Node Information Item 'dictItem' in [values], the corresponding keyMemberNode identified in the previous step MUST have a [values] that contains exactly one item. That item, referred to here as keyValue, MUST be assignable to [parent object][type][allowed key types], i.e., one of the following MUST be true:

- keyValue is a Text Node Information Item and [parent object][type][allowed key types] contains either the x:String XamlType Information Item (section 7.2.7), or the x:Object XamlType Information Item (section 7.2.6), or there is exactly one XamlMember Information Item in [parent object][type][allowed key types] with a non-Null [text syntax].
- keyValue is an Object Node Information Item, and [parent object][type][allowed key types] contains a keyType for which isAssignableTo(keyValue[type], keyType) is True

- Each keyValue identified in the previous step MUST be unique within the scope of the containing dictionary.

6.3.1.5 XML Data Rules

If [member][value type][is xdata] = True, then:

- If [values] contains more than 0 items, then:
  - [values] MUST NOT contain more than one item.
  - The one item in [values] MUST be a text node.
  - The value of the text node MUST be well-formed XML, and any namespace prefixes used within the XML MUST either be declared within the nested XML text, or be in the [xml namespace mappings] of the containing Member Node Information Item.

6.3.1.6 x:Class Directive Rules

If [member] is the x:Class Directive (section 7.3.7), [parent object][parent member] MUST be Null (i.e., this attribute may only be applied to the root node).

6.3.1.7 x:Subclass Directive Rules

If [member] is the x:Subclass Directive (section 7.3.8), [parent object][member nodes] MUST contain a Member Node Information Item whose [member] is the x:Class Directive (section 7.3.7).

6.3.1.8 x:ClassModifier Directive Rules

If [member] is the x:ClassModifier Directive (section 7.3.9), [parent object][member nodes] MUST contain a Member Node Information Item whose [member] is the x:Class Directive (section 7.3.7).

6.3.1.9 x:TypeArguments Directive Rules

If [member] is the x:TypeArguments Directive (section 7.3.11), [parent object][type][is generic] must be true.

6.3.1.10 x:FieldModifier Directive Rules

If [member] is the x:FieldModifier Directive (section 7.3.10), the root node's [member nodes] MUST contain a Member Node Information Item whose [member] is the x:Class Directive (section 7.3.7).

6.3.2 Validity Constraints

A Member Node Information Item in a Xaml Information Set cannot be valid with respect to some set of schemas unless it meets rules defined in this section.

6.3.2.1 Values Must Be of the Appropriate Type

For each item val in [values], one of the following MUST apply.

- val is a Text Node Information Item and
  - Either [member] is x:Items (section 7.3.1) and [parent object][type][allowed types] contains either the x:String (section 7.2.7), or the x:Object (section 7.2.6)XamlType Information Items.
  - or [member][value type] is either the x:String (section 7.2.7), or the x:Object (section 7.2.6)XamlType Information Item.
  - or [member][text syntax] is not Null.
  - or [member][value type][text syntax] is not Null.
- val is an Object Node Information Item and [member] is x:Items (section 7.3.1) and [member][value type][allowed types] contains an allowedType for which isAssignableTo (val[type], allowedType) is True.
- val is an Object Node Information Item where val[type][types assignable to] contains the intrinsic x:MarkupExtension (section 7.2.22), and [member] is x:Items (section 7.3.1) and [member][value type][allowed types] contains an allowedType for which isAssignableTo(val[type][return value type], allowedType) is True.
- val is an Object Node Information Item and isAssignableTo(val[type], [member][value type]) is True.
- val is an Object Node Information Item where val[type][types assignable to] contains the intrinsic x:MarkupExtension (section 7.2.22), and isAssignableTo(val[type][return value type], [member][value type]) is True.

6.3.2.2 If Member Non-Attached, Non-Directive, Element Type Must Have Member

If [member][is attachable] and [member][is directive] are both False, [parent object][type][members] MUST contain [member].

6.3.2.3 Attached Member Target Type Must Match

If [member][is attachable] is True, and [parent object][type][members] does not contain [member], isAssignableTo([parent object][type], [member][target type]) MUST be True.

6.3.2.4 Text Value of Non-Text Member Must Match Text Syntax

If [member][text syntax] is not Null, let textSyntax be [member][text syntax]; otherwise let textSyntax be [member][value type][text syntax].

If [values] contains a single Text Node Information Item, and textSyntax is not Null, the [text] value in the Text Node Information Item MUST either match one of the entries in textSyntax[values], or match one of the entries in textSyntax[patterns]. The rules for determining a match are defined in section 6.2.2.5 ("Initialization Text Must Match Text Syntax").

6.3.2.5 Read-Only Members

If [member][is read only] is True, [values] MUST contain just a single object node where [is retrieved] is True.

6.3.2.6 Names MUST Be Unique Within a Namespace Scope

Let nameMember of any Object Node Information Item be determined as follows:

- If that Object Node Information Item's [member nodes] contains an item whose [member] is the intrinsic x:Name Directive (section 7.3.4), nameMember is that item.
- If that Object Node Information Item's [member nodes] contains an item whose [member] is the Object Node Information Item's [parent object][type][name property], nameMember is that item.

If an Object Node Information Item has a nameMember, it is a 'named object', and its name is the single value in nameMember[values].

If an Object Node Information Item is a named object, its name MUST be different from the name of any other named object that shares the same namespace scope.

A Xaml processor MUST determine whether two Object Node Information Items share the same namespace scope by walking up the tree by starting at [parent object] and then following [parent object][parent member] until reaching either the root node, or an Object Node Information Item whose [type][is name scope] is True. If this process ends at the same node for two different nodes, those two nodes share a namespace scope.

6.3.2.7 x:Key Directive Rules

If [member] is the x:Key Directive (section 7.3.5), [parent object][parent member][parent object][type][is dictionary] MUST be True.

6.3.2.8 x:FieldModifier Directive Rules

If [member] is the x:FieldModifier Directive (section 7.3.10), [parent object][member nodes] MUST contain either a Member Node Information Item whose [member] is the x:Name Directive (section 7.3.4), or a Member Node Information Item whose [member] is the same as this member's [parent object][type][name property].

6.3.2.9 Members of Type x:XamlType and Type Names Must Refer to Valid Type

If [member][value type] is the intrinsic x:XamlType (section 7.2.20), or if [member] is the intrinsic TypeExtension.TypeName (section 7.3.22), or ([member] is the intrinsic x:PositionalParameters (section 7.3.2) AND [parent object][type] is the intrinsic x:TypeExtension (section 7.2.3)), then [values] MUST contain just a single text node. Let typeText be that text node's [text].

Let typeSchema be defined as follows:

- If typeText is a PrefixedName (as defined in section 4 of [XML Namespaces]), then:

This node's [xml namespace mappings] MUST contain an XML namespace mapping whose [prefix] matches the typeText's Prefix (as defined in section 4 of [XML Namespaces]); let nsUri be the [uri] of that XML namespace mapping.

- A schema whose [target namespace] matches nsUri MUST be available. Let memberSchema be the schema.
- Otherwise, [parent object][xml namespace mappings] MUST contain a mapping whose [prefix] is the empty string, and a schema MUST be available whose [target namespace] matches the [namespace name] of that mapping. Let memberSchema be that schema.

memberSchema[types] MUST contain a XamlType Information Item whose [name] matches typeText's LocalName (as defined in section 4 of [XML Namespaces]) in MemberText. Let targetType be that XamlType Information Item.

6.3.3 Notes (Non-Normative)

Validation rule 6.3.2.5 ("Read-only Members") deals with the fact that this is meaningful:

```
<MyElement>
  <MyElement.DictionaryMember>
    <AnotherElement x:Key="myKey" />
  </MyElement.DictionaryMember>
</MyElement>
```

As is this:

```
<MyElement>
  <MyElement.DictionaryMember>
    <DictionaryElement>
      <AnotherElement x:Key="myKey" />
    </DictionaryElement>
  </MyElement.DictionaryMember>
</MyElement>
```

But this is not:

```
<MyElement>
  <MyElement.DictionaryMember>
    <DictionaryElement>
      <AnotherElement x:Key="myKey" />
    </DictionaryElement>
    <AnotherElement x:Key="bar" />
  </MyElement.DictionaryMember>
</MyElement>
```

In summary, either you can bring your own dictionary, or you can add items directly to the member element as children to have them added to the dictionary, but you can't do both.

6.4 Text Node Information Item

A Text Node Information Item represents a value in a member node's [values] that contains text or an XML literal.

Table 14: Text Node Information Item Properties

Name	Type	Purpose (non-normative)
[text]	Text String	The text value.
[parent member]	Member Node Information Item	The member of which this text is a value.

6.4.1 Constraints

The Text Node Information Items in a well-formed Xaml Information Set MUST conform to the rules defined in this section.

6.4.1.1 Information Set Properties Must Be of Correct Type

Each property of a Text Node Information Item MUST have a value of the type specified for that property in Table 14.

6.4.2 Notes (Non-Normative)

Whitespace is represented as a text node. There are special processing rules for handling whitespace, because it is significant in some contexts, and ignorable in others. The Xaml Information Set is the output of the process described in section 8, "Creating a Xaml Information Set from XML", and all the rules for deciding whether to ignore whitespace are executed during that process. Consequently, there is no need to represent whitespace differently in the Xaml infoset--where whitespace is preserved, it simply ends up inside text nodes.

XML literals (xData) are also represented as text nodes. It is possible to determine whether a particular text node represents text or an XML literal by examining the text node's [parent member][member][value type][is xdata] member. Note that XML literals are not self-contained XML documents: the literal can use the namespace prefixes described in [parent member][xml namespace mappings], and they also inherit xml:space and xml:lang attributes where present.

Note that the set of characters that may be used in [text] is not restricted to the subset defined for the Char production defined in Section 2.2 of the XML specification ([XML]). However, the use of characters outside the range acceptable in XML is discouraged.

7 Intrinsic Schema Information Items

This section defines an intrinsic set of schema information items that are implicitly available to all Xaml Schema Information Sets and Xaml Information Sets.

7.1 Intrinsic Schema Information Items

The following Schema Information Items are available when processing any XML document as Xaml.

7.1.1 The 'x:' Schema

The 'x:' schema is so-called because its XML namespace is conventionally associated with the x: prefix. It contains directives and types available across all Xaml vocabularies.

Table 15: Schema Information Item Property Values for ‘x:’ Schema

Property	Value
[target namespace]	"http://schemas.microsoft.com/winfx/2006/xaml"
[types]	All of the XamlType Information Items defined in "Intrinsic XamlType Information Items" (section 7.2) of this specification.
[directives]	All of the XamlMember Information Items defined in "Intrinsic XamlMember Information Items" (section 7.3) of this specification.
[compatible with schemas]	Empty

7.1.2 The XML Namespace Schema

The XML Namespace Schema contains directives corresponding to the xml:lang, xml:space, and xml:base attributes.

Table 16: Schema Information Item Property Values for ‘x:’ Schema

Property	Value
[target namespace]	"http://www.w3.org/XML/1998/namespace"
[types]	Empty
[directives]	The following XamlMember Information Items: xml:lang Directive (section 7.3.13), xml:space Directive (section 7.3.14), and xml:base Directive (section 7.3.15)
[compatible with schemas]	Empty

7.2 Intrinsic XamlType Information Items

The following sections define special types used for processing Xaml Instances.

Many of the XamlType Information Item properties are somewhat specialized--the majority of types use the same values for most of the properties. To make it easy to see the distinguishing features of

each type, only the properties that differ from the norm are listed. Table 17 shows the values that properties have in the case where their type definition does not specify a value.

Table 17: Default XamlType Information Item Property Values

Property	Value
[types assignable to]	x:Object (section 7.2.6)
[is default constructible]	True
[is nullable]	True
[text syntax]	Null
[content property]	Null
[dictionary key property]	Null
[name property]	Null
[xml lang property]	Null
[trim surrounding whitespace]	False
[is whitespace significant collection]	False
[is list]	False
[is dictionary]	False
[allowed types]	Empty
[allowed key types]	Empty
[is xdata]	False
[is name scope]	False
[constructors]	Empty
[return value type]	Null
[is generic]	False

7.2.1 x:ArrayExtension

This XamlType Information Item signifies that a list of items is to be treated as an array, where the target programming system recognizes such a concept.

Table 18: XamlType Information Item Property Values for x:ArrayExtension

Property	Value
[name]	"ArrayExtension"
[members]	ArrayExtension.Items (section 7.3.18); ArrayExtension.Type (section 7.3.19)
[content property]	ArrayExtension.Items (section 7.3.18)

Note The distinction between this and x:Array (section 7.2.19) is that x:ArrayExtension is concerned with how to represent arrays in Xaml, whereas x:Array is the type of an array itself.

7.2.2 x:StaticExtension

This XamlType Information Item represents a markup extension type indicating that a static member belonging to some particular type should be read.

Table 19: XamlType Information Item Property Values for x:StaticExtension

Property	Value
[name]	"StaticExtension"
[types assignable to]	x:MarkupExtension (section 7.2.22); x:Object (section 7.2.6)
[members]	StaticExtension.Member (section 7.3.20)
[constructors]	Static Extension String Constructor (section 7.5.1)
[return value type]	x:Object (section 7.2.6)

7.2.3 x:TypeExtension

This XamlType Information Item represents a markup extension type that evaluates to a x:XamlType (section 7.2.20) object representing a particular type.

Table 20: XamlType Information Item Property Values for x:TypeExtension

Property	Value
[name]	"TypeExtension"
[types assignable to]	x:MarkupExtension (section 7.2.22); x:Object (section 7.2.6)
[members]	TypeExtension.Type (section 7.3.21); TypeExtension.TypeName (section 7.3.22)
[constructors]	Type Extension String Constructor (section 7.5.2)
[return value type]	x:XamlType (section 7.2.20)

7.2.4 x:NullExtension

This XamlType Information Item represents a markup extension type that evaluates to a Null value.

Table 21: XamlType Information Item Property Values for x:NullExtension

Property	Value
[name]	"NullExtension"
[types assignable to]	x:MarkupExtension (section 7.2.22); x:Object (section 7.2.6)
[members]	Empty

Property	Value
[return value type]	x:Object (section 7.2.6)

7.2.5 x:ReferenceExtension

This XamlType Information Item represents a markup extension type that represents a reference to a named Object Node instance.

Table 22: XamlType Information Item Property Values for x:ReferenceExtension

Property	Value
[name]	"ReferenceExtension"
[types assignable to]	x:MarkupExtension (section 7.2.22); x:Object (section 7.2.6)
[members]	ReferenceExtension.Name (section 7.3.23)
[constructors]	Reference Extension String Constructor (section 7.5.3)
[return value type]	x:Object (section 7.2.6)

7.2.6 x:Object

Xaml vocabularies that wish to be able to define members that can accept values of any type can use x:Object as the [value type] of those members. Note that this specification does not assign any intrinsic meaning to x:Object--it only provides it as a common type for vocabularies that require this idea. Since Xaml does not require vocabularies to support a 'root type' or 'universal base class' idea, those that want this concept must opt in: a type will only be assignable to x:Object if the vocabulary explicitly includes x:Object in that type's [types assignable to].

Table 23: XamlType Information Item Property Values for x:Object

Property	Value
[name]	"Object"
[types assignable to]	Empty
[is default constructible]	False
[members]	Empty

7.2.7 x:String

Type used to represent text strings.

Table 24: XamlType Information Item Property Values for x:String

Property	Value
[name]	"String"
[members]	Empty

7.2.8 x:Char

Type used to represent a single text character.

Table 25: XamlType Information Item Property Values for x:Char

Property	Value
[name]	"Char"
[is nullable]	False
[text syntax]	x:Char Text Syntax (section 7.4.1)
[members]	Empty

7.2.9 x:Single

Type representing a single-precision floating point numeric value. This type is not used within this specification. It is provided because this type is widely used--by providing this one definition, Xaml vocabularies do not each need to define their own.

Table 26: XamlType Information Item Property Values for x:Single

Property	Value
[name]	"Single"
[is nullable]	False
[text syntax]	x:Single Text Syntax (section 7.4.2)
[members]	Empty

7.2.10 x:Double

Type representing a double-precision floating point numeric value. This type is not used within this specification. It is provided because this type is widely used--by providing this one definition, Xaml vocabularies do not each need to define their own.

Table 27: XamlType Information Item Property Values for x:Double

Property	Value
[name]	"Double"
[is nullable]	False
[text syntax]	x:Double Text Syntax (section 7.4.3)
[members]	Empty

7.2.11 x:Byte

Type representing an unsigned 8-bit byte. This type is not used within this specification. It is provided because this type is widely used - by providing this one definition, Xaml vocabularies do not each need to define their own.

Table 28: XamlType Information Item Property Values for x:Byte

Property	Value
[name]	"Byte"
[is nullable]	False
[text syntax]	x:Byte Text Syntax (section 7.4.4)
[members]	Empty

7.2.12 x:Int16

Type representing a signed 16-bit number. This type is not used within this specification. It is provided because this type is widely used - by providing this one definition, Xaml vocabularies do not each need to define their own.

Table 29: XamlType Information Item Property Values for x:Int16

Property	Value
[name]	"Int16"
[is nullable]	False
[text syntax]	x:Int16 Text Syntax (section 7.4.5)
[members]	Empty

7.2.13 x:Int32

Type representing a signed 32-bit number. This type is not used within this specification. It is provided because this type is widely used - by providing this one definition, Xaml vocabularies do not each need to define their own.

Table 30: XamlType Information Item Property Values for x:Int32

Property	Value
[name]	"Int32"
[is nullable]	False
[text syntax]	x:Int32 Text Syntax (section 7.4.6)
[members]	Empty

7.2.14 x:Int64

Type representing a signed 64-bit number. This type is not used within this specification. It is provided because this type is widely used - by providing this one definition, Xaml vocabularies do not each need to define their own.

Table 31: XamlType Information Item Property Values for x:Int64

Property	Value
[name]	"Int64"
[is nullable]	False
[text syntax]	x:Int64 Text Syntax (section 7.4.7)
[members]	Empty

7.2.15 x:Decimal

Type representing a decimal number. This type is not used within this specification. It is provided because this type is widely used - by providing this one definition, Xaml vocabularies do not each need to define their own.

Table 32: XamlType Information Item Property Values for x:Decimal

Property	Value
[name]	"Decimal"
[is nullable]	False
[text syntax]	x:Decimal Text Syntax (section 7.4.8)
[members]	Empty

7.2.16 x:Uri

Type representing a URI. This type is not used within this specification. It is provided because this type is widely used - by providing this one definition, Xaml vocabularies do not each need to define their own.

Table 33: XamlType Information Item Property Values for x:Uri

Property	Value
[name]	"Uri"
[is nullable]	False
[text syntax]	x:Uri Text Syntax (section 7.4.9)
[members]	Empty

7.2.17 x:Timespan

Type representing a time span. This type is not used within this specification. It is provided because this type is widely used - by providing this one definition, Xaml vocabularies do not each need to define their own.

Table 34: XamlType Information Item Property Values for x:Timespan

Property	Value
[name]	"Timespan"
[is nullable]	False
[text syntax]	x:Timespan Text Syntax (section 7.4.10)
[members]	Empty

7.2.18 x:Boolean

Type representing a Boolean value--a value which may be either true or false.

Table 35: XamlType Information Item Property Values for x:Boolean

Property	Value
[name]	"Boolean"
[is nullable]	False
[text syntax]	x:Boolean Text Syntax (section 7.4.11)
[members]	Empty

7.2.19 x:Array

Type representing base class of arrays.

Table 36: XamlType Information Item Property Values for x:Array

Property	Value
[name]	"Array"
[is default constructible]	False
[members]	Empty
[is list]	True
[allowed types]	x:Object (section 7.2.6)

7.2.20 x:XamlType

Type for objects that represent types.

Table 37: XamlType Information Item Property Values for x:XamlType

Property	Value
[name]	"XamlType"
[is default constructible]	False
[text syntax]	x:XamlType Text Syntax (section 7.4.12)
[members]	Empty
[return value type]	Null

7.2.21 x:XamlEvent

Type for objects that represent event handlers.

Table 38: XamlType Information Item Property Values for x:XamlEvent

Property	Value
[name]	"XamlEvent"
[is default constructible]	False
[text syntax]	x:XamlEvent Text Syntax (section 7.4.14)
[members]	Empty

7.2.22 x:MarkupExtension

Well-known base type that indicates a type is a markup extension. Types indicate that they are markup extensions by including this in their [types assignable to].

Table 39: XamlType Information Item Property Values for x:MarkupExtension

Property	Value
[name]	"MarkupExtension"
[is default constructible]	False
[members]	Empty

7.2.23 x:Code

Type of object nodes representing source code embedded in Xaml. (The meaning of this code is determined by the Xaml processor. This specification simply defines the mechanism by which plain text can be embedded in a Xaml Instance.)

Table 40: XamlType Information Item Property Values for x:Code

Property	Value
[name]	"Code"
[is default constructible]	False
[is nullable]	False
[members]	Empty

7.2.24 x:XData

Type representing XML data islands.

Table 41: XamlType Information Item Property Values for x:XData

Property	Value
[name]	"XData"
[is default constructible]	False
[members]	Empty

7.3 Intrinsic XamlMember Information Items

This section defines the intrinsic members defined in the x: Schema and the XML Namespace Schema.

Many of the XamlMember Information Item properties are somewhat specialized - the majority of members use the same values for some properties. To make it easy to see the distinguishing features of each member, only properties that differ from the norm are listed. Table 42 shows the values that properties have in the case where their type definition does not specify a value.

Table 42: Default XamlMember Information Item Property Values

Property	Value
[text syntax]	Null
[is read only]	False
[is static]	False
[is attachable]	False
[target type]	Null
[allowed location]	Any
[is event]	False
[is directive]	False

7.3.1 x:Items

Pseudo member used when an object node contains multiple values.

Table 43 XamlMember Information Item Property Values for x:Items

Property	Value
[name]	"Items"
[owner type]	Null
[value type]	x:Object (section 7.2.6)
[allowed location]	None
[is directive]	True

7.3.2 x:PositionalParameters

Pseudo member used to represent the constructor argument list for a markup extension. The markup extension's object node's [member nodes] will contain a member for each named parameter, and may also contain a member to represent the unnamed, ordered parameters. A member node serving that purpose is identified by having its [member] refer to this x:PositionalParameters intrinsic member.

Table 44 XamlMember Information Item Property Values for x:PositionalParameters

Property	Value
[name]	"PositionalParameters"
[owner type]	Null
[value type]	x:String (section 7.2.7)
[allowed location]	None
[is directive]	True

7.3.3 x:Initialization

Pseudo member used to hold the text string used when an object is initialized using text.

Table 45 XamlMember Information Item Property Values for x:Initialization

Property	Value
[name]	"Initialization"
[owner type]	Null
[value type]	x:String (section 7.2.7)
[allowed location]	None
[is directive]	True

7.3.4 x:Name Directive

Directive used to set the name of an element. The meaning of a 'name' is up to individual Xaml processors to describe - this specification simply provides a mechanism for associating a name with an object node, and a mechanism by which a schema can indicate that a particular member is equivalent to the x:Name directive.

Table 46 XamlMember Information Item Property Values for x:Name Directive

Property	Value
[name]	"Name"
[owner type]	Null
[value type]	x:String (section 7.2.7)
[is directive]	True

7.3.5 x:Key Directive

Directive used to indicate the key of an object added to a dictionary.

Table 47 XamlMember Information Item Property Values for x:Key Directive

Property	Value
[name]	"Key"
[owner type]	Null
[value type]	x:Object (section 7.2.6)
[is directive]	True

7.3.6 x:Uid Directive

Directive used to provide a unique identifier for localization purposes.

Table 48 XamlMember Information Item Property Values for x:Uid Directive

Property	Value
[name]	"Uid"
[owner type]	Null
[value type]	x:String (section 7.2.7)
[is directive]	True

7.3.7 x:Class Directive

Directive used to indicate the name of a class associated with a Xaml file. It is up to individual Xaml processors to define the interpretation of this directive.

Table 49 XamlMember Information Item Property Values for x:Class Directive

Property	Value
[name]	"Class"
[owner type]	Null
[value type]	x:String (section 7.2.7)
[is directive]	True

7.3.8 x:Subclass Directive

Directive used to indicate the name of a subclass associated with a Xaml file. It is up to individual Xaml processors to define the interpretation of this directive.

Table 50 XamlMember Information Item Property Values for x:Subclass Directive

Property	Value
[name]	"Subclass"
[owner type]	Null
[value type]	x:String (section 7.2.7)
[is directive]	True

7.3.9 x:ClassModifier Directive

Directive used to indicate the modifier of the class associated with a Xaml file. It is up to individual Xaml processors to define the interpretation of this directive.

Table 51 XamlMember Information Item Property Values for x:ClassModifier Directive

Property	Value
[name]	"ClassModifier"
[owner type]	Null
[value type]	x:String (section 7.2.7)
[is directive]	True

7.3.10 x:FieldModifier Directive

Directive used to indicate the modifier of the field associated with a named Xaml element. It is up to individual Xaml processors to define the interpretation of this directive.

Table 52 XamlMember Information Item Property Values for x:FieldModifier Directive

Property	Value
[name]	"FieldModifier"
[owner type]	Null
[value type]	x:String (section 7.2.7)
[is directive]	True

7.3.11 x:TypeArguments Directive

Directive used to indicate the type arguments for the class associated with a Xaml file. It is up to individual Xaml processors to define the interpretation of this directive.

Table 53 XamlMember Information Item Property Values for x:TypeArguments Directive

Property	Value
[name]	"TypeArguments"
[owner type]	Null
[value type]	x:String (section 7.2.7)
[allowed location]	AttributeOnly
[is directive]	True
[text syntax]	x:TypeArguments Text Syntax (section 7.4.16)

7.3.12 x:DirectiveChildren

Pseudo member used to hold x:Code items.

Table 54 XamlMember Information Item Property Values for x:DirectiveChildren

Property	Value
[name]	"DirectiveChildren"
[owner type]	Null
[value type]	x:Code (section 7.2.23)
[allowed location]	None
[is directive]	True

7.3.13 xml:lang Directive

Directive used to represent the standard xml:lang attribute in a Xaml file.

Table 55 XamlMember Information Item Property Values for xml:lang Directive

Property	Value
[name]	"TypeArguments"
[owner type]	Null
[value type]	x:String (section 7.2.7)
[allowed location]	AttributeOnly
[is directive]	True

7.3.14 xml:space Directive

Directive used to represent the standard xml:space attribute in a Xaml file.

Table 56 XamlMember Information Item Property Values for xml:space Directive

Property	Value
[name]	"space"
[owner type]	Null
[value type]	x:String (section 7.2.7)
[text syntax]	xml:space Text Syntax (section 7.4.13)
[allowed location]	AttributeOnly
[is directive]	True

7.3.15 xml:base Directive

Directive used to represent the standard xml:base attribute in a Xaml file.

Table 57 XamlMember Information Item Property Values for xml:base Directive

Property	Value
[name]	"base"
[owner type]	Null
[value type]	x:String (section 7.2.7)
[allowed location]	AttributeOnly
[is directive]	True

7.3.16 x:Arguments Directive

Directive used to represent a list of arguments for the object.

Table 58: XamlMember Information Item Property Values for x:Arguments Directive

Property	Value
[name]	"Arguments"
[owner type]	Null
[value type]	x:Object (section 7.2.6)
[allowed location]	InitialMemberElementsOnly
[is directive]	True

7.3.17 x:FactoryMethod Directive

Directive used to represent a factory method for the object.

Table 59 XamlMember Information Item Property Values for x:FactoryMethod Directive

Property	Value
[name]	"FactoryMethod"
[owner type]	Null
[value type]	x:String (section 7.2.7)
[allowed location]	AttributeOrInitialMemberElementsOnly
[is directive]	True
[text syntax]	x:FactoryMethod Text Syntax (section 7.4.17)

7.3.18 ArrayExtension.Items

Content member for the x:ArrayExtension (section 7.2.1) type.

Table 60 XamlMember Information Item Property Values for ArrayExtension.Items

Property	Value
[name]	"Items"
[owner type]	x:ArrayExtension (section 7.2.1)
[value type]	x:Array (section 7.2.19)
[is read only]	True

7.3.19 ArrayExtension.Type

Member of x:ArrayExtension (section 7.2.1) that indicates what type of elements the array contains.

Table 61 XamlMember Information Item Property Values for ArrayExtension.Type

Property	Value
[name]	"Type"
[owner type]	x:ArrayExtension (section 7.2.1)
[value type]	x:XamlType (section 7.2.20)

7.3.20 StaticExtension.Member

Identifies the static member whose value this extension returns.

Table 62 XamlMember Information Item Property Values for StaticExtension.Member

Property	Value
[name]	"Member"
[owner type]	x:StaticExtension (section 7.2.2)
[value type]	x:String (section 7.2.7)

7.3.21 TypeExtension.Type

The XamlType this extension returns.

Table 63 XamlMember Information Item Property Values for TypeExtension.Type

Property	Value
[name]	"Type"
[owner type]	x:TypeExtension (section 7.2.3)
[value type]	x:XamlType (section 7.2.20)

7.3.22 TypeExtension.TypeName

The name of the XamlType this extension returns.

Table 64 XamlMember Information Item Property Values for TypeExtension.TypeName

Property	Value
[name]	"TypeName"
[owner type]	x:TypeExtension (section 7.2.3)
[value type]	x:String (section 7.2.7)
[text syntax]	x:XamlType Text Syntax (section 7.4.12)

7.3.23 ReferenceExtension.Name

The name of the object this extension references. A reference should be treated as if the referenced object was in its place.

Table 65 XamlMember Information Item Property Values for ReferenceExtension.Name

Property	Value
[name]	"Name"
[owner type]	x:TypeExtension (section 7.2.3)
[value type]	x:String (section 7.2.7)
[text syntax]	x:NameReference Text Syntax (section 7.4.15)

7.4 Intrinsic Text Syntax Information Items

This section defines the intrinsic Text Syntax Information Items.

This section uses a notational convention for representing the Value Syntax Information Items and Pattern Syntax Information Items that make up a Text Syntax Information Item. They are listed as a semicolon delimited sequence, or the word Empty to indicate that the relevant set contains no items.

Each entry in the sequence for the [values] of a Text Syntax Information Item represents a Value Syntax Information Item. The entry includes a string in double quotes representing the [text] member of the Value Syntax Information Item. The string may optionally be followed by [is case sensitive]=True, and/or [trim whitespace]=False. Where these values are not specified explicitly, [is case sensitive] is False, and [trim whitespace] is True.

Each entry in the sequence for the [patterns] of a Text Syntax Information Item represents a Pattern Syntax Information Item. The entry includes a string in double quotes representing the [pattern] member of the Pattern Syntax Information Item. This string may optionall be followed by [is case sensitive]=False, and/or [trim whitespace]=False. Where these values are not specified explicitly, [is case sensitive] is True, and [trim whitespace] is True.

7.4.1 x:Char Text Syntax

This text syntax defines the acceptable representations of values of type x:Char (section 7.2.8).

Table 66 Text Syntax Information Item Property Values for x:Char Text Syntax

Property	Value
[values]	Empty
[patterns]	"."

7.4.2 x:Single Text Syntax

This text syntax defines the acceptable representations of values of type x:Single (section 7.2.9).

Table 67 Text Syntax Information Item Property Values for x:Single Text Syntax

Property	Value
[values]	"Infinity" [is case sensitive]=True; "-Infinity" [is case sensitive]=True; "NaN" [is case sensitive]=True
[patterns]	"[+-]?(([\d,]+(\.\d*)?)|([\d,]*\.\d+))([eE][+-]?\d+)?"

Informally, the [patterns] allows optional leading whitespace followed by an optional + or - sign, followed by a decimal number that takes one of two forms, optionally followed by an exponent. The first acceptable form for the decimal number is a sequence of one or more decimal digits, optionally followed by a decimal point which is followed by a sequence of zero or more decimal digits. (E.g. "1" or "1.01".) The second acceptable form for the decimal number is a sequence of zero or more decimal digits, followed by a decimal point which is followed by a sequence of one or more decimal digits. (E.g. ".1" or "1.01".) The optional exponent must start with either 'e' or 'E', optionally followed by a + or - sign, and then followed by a sequence of one or more decimal digits. Finally, the number value may be followed by whitespace.

7.4.3 x:Double Text Syntax

This text syntax defines the acceptable representations of values of type x:Double (section 7.2.10).

Table 68 Text Syntax Information Item Property Values for x:Double Text Syntax

Property	Value
[values]	"Infinity" [is case sensitive]=True; "-Infinity" [is case sensitive]=True; "NaN" [is case sensitive]=True
[patterns]	"[+-]?(([\d,]+(\.\d*)?)|([\d,]*\.\d+))([eE][+-]?\d+)?"

Informally, the [patterns] allows optional leading whitespace followed by an optional + or - sign, followed by a decimal number that takes one of two forms, optionally followed by an exponent. The first acceptable form for the decimal number is a sequence of one or more decimal digits, optionally followed by a decimal point which is followed by a sequence of zero or more decimal digits. (E.g. "1" or "1.01".) The second acceptable form for the decimal number is a sequence of zero or more decimal digits, followed by a decimal point which is followed by a sequence of one or more decimal digits. (E.g. ".1" or "1.01".) The optional exponent must start with either 'e' or 'E', optionally followed by a + or - sign, and then followed by a sequence of one or more decimal digits. Finally, the number value may be followed by whitespace.

7.4.4 x:Byte Text Syntax

This text syntax defines the acceptable representations of values of type x:Byte (section 7.2.11).

Table 69: Text Syntax Information Item Property Values for x:Byte Text Syntax

Property	Value
[values]	Empty
[patterns]	"\d+"

7.4.5 x:Int16 Text Syntax

This text syntax defines the acceptable representations of values of type x:Int16 (section 7.2.12).

Table 70 Text Syntax Information Item Property Values for x:Int16 Text Syntax

Property	Value
[values]	Empty
[patterns]	"[+-]?\d+"

7.4.6 x:Int32 Text Syntax

This text syntax defines the acceptable representations of values of type x:Int32 (section 7.2.13).

Table 71 Text Syntax Information Item Property Values for x:Int32 Text Syntax

Property	Value
[values]	Empty
[patterns]	"[+-]?\d+"

7.4.7 x:Int64 Text Syntax

This text syntax defines the acceptable representations of values of type x:Int64 (section 7.2.14).

Table 72 Text Syntax Information Item Property Values for x:Int64 Text Syntax

Property	Value
[values]	Empty
[patterns]	"[+-]?\d+"

7.4.8 x:Decimal Text Syntax

This text syntax defines the acceptable representations of values of type x:Decimal (section 7.2.15).

Table 73 Text Syntax Information Item Property Values for x:Decimal Text Syntax

Property	Value
[values]	Empty
[patterns]	"[+-]?\d+"

7.4.9 x:Uri Text Syntax

This text syntax defines the acceptable representations of values of type x:Uri (section 7.2.16).

Table 74 Text Syntax Information Item Property Values for x:Uri Text Syntax

Property	Value
[values]	Empty
[patterns]	"*"

7.4.10 x:Timespan Text Syntax

This text syntax defines the acceptable representations of values of type x:Timespan (section 7.2.17).

Table 75 Text Syntax Information Item Property Values for x:Timespan Text Syntax

Property	Value
[values]	Empty
[patterns]	"-?(\d*\.)?\d\d?:\d\d?:((\d\d?)|(\d?\d?\.\d*))"; "-?\d+"

7.4.11 x:Boolean Text Syntax

This text syntax defines the acceptable representations of values of type x:Boolean (section 7.2.18).

Table 76 Text Syntax Information Item Property Values for x:Boolean Text Syntax

Property	Value
[values]	"True"; "False"
[patterns]	Empty

Informally, the [patterns] allows either the string "true" or "false" (with any mixture of upper or lower case characters), optionally surrounded by whitespace.

7.4.12 x:XamlType Text Syntax

This text syntax defines the acceptable representations of values of type x:XamlType (section 7.2.20).

Table 77 Text Syntax Information Item Property Values for x:XamlType Text Syntax

Property	Value
[values]	Empty
[patterns]	"([_\p{L}][\.-_\p{L}\p{Nd}\p{Mc}]*:)?[_\p{Lu}\p{Ll}\p{Lo}\p{Lt}\p{Nl}\p{Lm}][_\p{Lu}\p{Ll}\p{Lo}\p{Lt}\p{Nl}\p{Lm}\p{Nd}\p{Mn}\p{Mc}]*" [trim whitespace]=False

Informally, the [patterns] allows an optional namespace prefix which, if present, must be followed by a colon, followed by a XamlName.

7.4.13 xml:space Text Syntax

This text syntax describes the acceptable representations of values of the xml:space Directive (section 7.3.14).

Table 78 Text Syntax Information Item Property Values for xml:space Text Syntax

Property	Value
[values]	"default" [is case sensitive]=True [trim whitespace]=False; "preserve" [is case sensitive]=True [trim whitespace]=False
[patterns]	Empty

7.4.14 x:XamlEvent Text Syntax

This text syntax describes the acceptable representations of event members.

Table 79 Text Syntax Information Item Property Values for x:XamlEvent Text Syntax

Property	Value
[values]	Empty
[patterns]	"[_\p{Lu}\p{Ll}\p{Lo}\p{Lt}\p{Nl}\p{Lm}][_\p{Lu}\p{Ll}\p{Lo}\p{Lt}\p{Nl}\p{Lm}\p{Nd}\p{Mn}\p{Mc}]*" [trim whitespace]=False

Informally, the [patterns] allows any XamlName.

7.4.15 x:NameReference Text Syntax

This text syntax describes the acceptable representations of values which are name references.

Table 80 Text Syntax Information Item Property Values for x:NameReference Text Syntax

Property	Value
[values]	Empty
[patterns]	"[_\p{Lu}\p{Ll}\p{Lo}\p{Lt}\p{Nl}\p{Lm}][_\p{Lu}\p{Ll}\p{Lo}\p{Lt}\p{Nl}\p{Lm}\p{Nd}\p{Mn}\p{Mc}]*" [trim whitespace]=False

Informally, the [patterns] allows any XamlName.

7.4.16 x:TypeArguments Text Syntax

This text syntax describes the acceptable representations of values which are type arguments.

Table 81 Text Syntax Information Item Property Values for x:TypeArguments Text Syntax

Property	Value
[values]	Empty

Property	Value
[patterns]	"[[_\p{Lu}\p{Ll}\p{Lo}\p{Lt}\p{Nl}\p{Lm}][_\p{Lu}\p{Ll}\p{Lo}\p{Lt}\p{Nl}\p{Lm}\p{Nd}\p{Mn}\p{Mc}]*[,[_\p{Lu}\p{Ll}\p{Lo}\p{Lt}\p{Nl}\p{Lm}][_\p{Lu}\p{Ll}\p{Lo}\p{Lt}\p{Nl}\p{Lm}\p{Nd}\p{Mn}\p{Mc}]*]" [trim whitespace]=False

Informally, the [patterns] allows a comma delimited ordered collection of XamlTypes, any of which could be closed generic types with its type arguments specified inside of parenthesis.

7.4.17 x:FactoryMethod Text Syntax

This text syntax describes the acceptable representations of values which are factory methods.

Table 82 Text Syntax Information Item Property Values for x:FactoryMethod Text Syntax

Property	Value
[values]	Empty
[patterns]	"[[_\p{Lu}\p{Ll}\p{Lo}\p{Lt}\p{Nl}\p{Lm}][_\p{Lu}\p{Ll}\p{Lo}\p{Lt}\p{Nl}\p{Lm}\p{Nd}\p{Mn}\p{Mc}].]?[_\p{Lu}\p{Ll}\p{Lo}\p{Lt}\p{Nl}\p{Lm}][_\p{Lu}\p{Ll}\p{Lo}\p{Lt}\p{Nl}\p{Lm}\p{Nd}\p{Mn}\p{Mc}]*" [trim whitespace]=False

Informally, the [patterns] allows any method name, optionally preceded with a XamlType name and a period.

7.5 Intrinsic Constructor Information Items

This section defines the Constructor Information Item used by the XamlType Information Items defined in section 7.2, "Intrinsic XamlType Information Items".

7.5.1 Static Extension String Constructor

This is the only Constructor Information Item for the x:StaticExtension type (section 7.2.2). It takes a single parameter, a string, which is logically equivalent to the StaticExtension.Member (section 7.3.20) member.

Table 83: Constructor Information Item Property Values for Static Extension String Constructor

Property	Value
[arguments]	[x:String (section 7.2.7)]

7.5.2 Type Extension String Constructor

This is the only Constructor Information Item for the x:TypeExtension type (Note (non-normative)). It takes a single parameter, a string, which is logically equivalent to the TypeExtension.TypeName (section 7.3.22) member.

Table 84: Constructor Information Item Property Values for x:TypeExtension

Property	Value
[arguments]	[x:String (section 7.2.7)]

7.5.3 Reference Extension String Constructor

This is the only Constructor Information Item for the x:ReferenceExtension type (section 7.2.5). It takes a single parameter, a string, which is logically equivalent to the ReferenceExtension.NameTypeExtension.TypeName (section 7.3.23) member.

Table 85: Constructor Information Item Property Values for x:ReferenceExtension

Property	Value
[arguments]	[x:String (section 7.2.7)]

8 Creating a Xaml Information Set from XML

An XML document is a Xaml Document if a well-formed Xaml Information Set can be created from the XML document's XML Infoset. The conversion process is performed with reference to a set of schemas that includes those defined in "Intrinsic Schema Information Items" (section 7). Additional vocabulary-specific schemas may (and usually will) also be used as part of the conversion process.

In order to construct a Xaml Information Set from an XML document, that document's XML Infoset is processed with the rules defined in section 8.6. If the rules in section 8.6are successfully executed without error, the resulting Xaml Information Set must conform to the constraints in Xaml Schema Information Set (section 5) in order for the XML document to be said to be a Xaml Document.

8.1 Unavailability of Xaml Schemas

XML documents may contain features for which schema information items are not available. This makes it impossible to determine that the document is valid. Implementations MAY choose to process documents despite the absence of schemas, producing a well-formed but potentially invalid Xaml Information Set.

Implementations that choose to handle Xaml in the absence of schema MUST generate placeholder Xaml Schema Information Set items to stand in for the missing schema. For example, a Member Node Information Item has a [member] property that refers to a XamlMember Information Item. Without this, there would be no way to know the member's name. Section 8.6 includes rules for generating suitable information items.

8.2 Processing Errors

Not all XML documents can be successfully converted to a Xaml Information Set. However, it may be useful for Xaml processors to continue processing in the face of an error. This specification does not define what Xaml processors should do in the face of an error - Xaml processors MAY terminate processing in the face of an error or they MAY attempt to continue processing.

Where the rules in this section identify an error, they incorporate a description. For example, a particular scenario might be described as an "Unknown namespace" error. The error text is informative, and Xaml processors are not required to do anything with that text.

8.3 Markup Compatibility

XML documents containing Xaml may use the Markup Compatibility and Extensibility conventions defined in Part 5 of [ECMA-376]. Implementations that convert XML into a Xaml Information Set can process documents in two modes with respect to markup compatibility: raw, and preprocessed. Xaml processors that support XML MUST support at least one of these modes.

8.3.1 Raw Mode

In raw mode, elements and attributes in the XML document from the markup compatibility namespace are processed in exactly the same way as any other elements and attributes, forming object, member, and text nodes as normal.

Raw mode is suitable for applications that need to preserve as much of the original document's structure as possible. A Xaml editor might use this mode.

8.3.2 Preprocessed Mode

In preprocessed mode, the input XML infoset is transformed by processing the content according to the rules defined in the ECMA Markup Compatibility Specification [ECMA-376]--content is ignored or substituted where appropriate. The markup compatibility elements and attributes are then removed. The resulting XML infoset becomes the input to the conversion process described in this section.

8.3.3 Subsumption Behavior

Section 10.2 of the Markup Compatibility specification requires markup consumers to declare which subsumption behavior is used when processing an element from an older namespace that carries a prefixed attribute from a newer, subsuming namespace. In a Xaml Document, if the attribute name contains a "." then the expanded name refers to the new namespace, if not, it refers to no namespace.

8.4 XML Information Set References

The XML processing rules take an XML Information Set [XML Infoset] as input. To indicate that a particular item is an XML Information Set information item, it is referred to as XML:*type*, where *type* is one of the XML Information Set information item types:

Table 86: XML Information Set Reference Names

Name in This Specification	Full Name in XML Information Set
XML:document	Document Information Item
XML:element	Element Information Item
XML:attribute	Attribute Information Item
XML:namespace	Namespace Information Item
XML:character	Character Information Item

8.5 Definitions

The following definitions are used in the conversion process.

8.5.1 DottedXamlName

A DottedXamlName is a string that conforms to the following grammar (using the syntax defined in [RFC4234]):

```
DottedXamlName = XamlName "." XamlName
```

The XamlName production was defined in section 4.2.

The first XamlName in a DottedXamlName is referred to as the typename. The second XamlName is referred to as the membername.

8.5.2 Collapsible Whitespace Characters

The characters subject to whitespace collapsing are the following three Unicode code points:

Table 87: Collapsible Whitespace Characters

Unicode Code Point	Character
0020	Space
000A	LineFeed
0009	Tab

Note that it is intentional that this is not a complete list of all whitespace Unicode code points. There are other Unicode codepoints that represent forms of whitespace, but those do not get collapsed.

8.5.3 Linefeed Collapsing Characters

The Unicode code points and surrogates in Table 88 and Table 89 are identified as 'Linefeed Collapsing characters', and are subject to special rules for whitespace collapsing:

Table 88: Linefeed Collapsing Code Points

Code Point Range (Inclusive)	Characters
1100 - 11FF	Hangul
2E80 - 2FD5	CJK and KangXi Radicals
2FF0 - 2FFB	Ideographic Description
3040 - 309F	Hiragana
30A0 - 30FF	Katakana
3100 - 312F	Bopomofo
3130 - 318F	Hangul Compatibility Jamo
3190 - 319F	Kanbun
31F0 - 31FF	Katakana Phonetic Extensions
3400 - 4DFF	CJK Unified Ideographs Extension A
4E00 - 9FFF	CJK Unified Ideographs
A000 - A4CF	Yi
AC00 - D7A3	Hangul Syllables
F900 - FAFF	CJK Compatibility
FF00 - FFEF	Halfwidth and Fullwidth Forms

Table 89: Linefeed Collapsing Surrogates

Surrogate Range (Inclusive)	Characters
20000 - 2A6D6	CJK Unified Ext. B

Surrogate Range (Inclusive)	Characters
2F800 - 2FA1D	CJK Compatibility Supplement

The special rules for these characters are defined later in this specification.

8.5.4 Authoritative Schema

For an XML:element e, if a Xaml Schema is available whose [target namespace] matches e[namespace name], that schema is the authoritative schema.

For an XML:attribute attr, the authoritative schema is determined as follows:

If attr[namespace name] has a value, one of the following two rules applies:

- If a schema is available whose [target namespace] matches the attribute's namespace, that schema is the authoritative schema.
- If no schema is available whose [target namespace] matches the attribute's namespace, this is an "Unknown namespace" error.

If attr[namespace name] does not have a value, one of the following rules applies:

- If the attr[local name] matches the XamlName production, the authoritative schema is the authoritative schema of the element to which the attribute has been applied.
- If the attr[local name] matches the DottedXamlName production:
  - If a schema is available whose [target namespace] matches the default namespace in scope, that is the authoritative schema.
  - If no schema is available whose [target namespace] matches the default namespace in scope, this is an "Unknown namespace" error.
- Otherwise, there is no authoritative schema.

The rules in 8.6 use the syntax "schema(node)" where node is either an XML:element or an XML:attribute. This is shorthand meaning the authoritative schema for either an element or an attribute node.

8.6 Document Processing Rules

This section defines rules for processing an XML Infoset to generate a Xaml Information Set. Each rule takes inputs that determine the context in which the rule is executed. Each rule has a single output. Most rules return a Xaml Information Set item, except for a few utility rules used to avoid duplication of logic. (E.g., some rules return a Xaml Schema Information Set item.)

Many of the rules involve several steps. These rules define variables to hold the intermediate results. For clarity, rule inputs and variables are shown with distinctive formatting, as shown in Table 90.

Table 90: Rule input and variable formatting

Example	Meaning
myInput	An input to a rule

Example	Meaning
myVariable	A variable whose value is the intermediate result of a step of a rule

Rules often define some steps in terms of other rules. The following formatting convention is used to denote an invocation of a rule:

```
Let resultVar = Invoke "Member Node Creation from Content"
                                    (containingMember ::= contentMember,
                                    memberType ::= memberType,
                                    childNodes ::= npChildren,
                                    preserveXmlSpace ::= space)
```

This signifies that a rule variable called *resultVar* should be given the value returned by invoking the rule named "Member Node Creation from Content", passing various rule variables in as the inputs to the rule.

Processing MUST begin by passing the XML:document information item of the information set of the XML document to be processed as the *xmlDocument* input to the 'XML:document Processing' rule defined in 8.6.1.

8.6.1 XML:document Processing

This rule has the following input:

Name	Type	Purpose (non-normative)
xmlDocument	XML:document	The XML document to be processed.

The output of this rule is a Xaml Information Set Document Information Item.

If *xmlDocument*[children] contains a document type definition (DTD), it is a "Xaml documents must not contain DTDs" error.

If the document contains entity references other than lt, gt, amp, apos, or quot, it is a "Xaml documents must not contain entity references other than lt, gt, amp, apos, or quot" error. This rule also implies that *xmlDocument*[notations] and *xmlDocument*[unparsed entities] must be empty.

A Xaml Document processor MUST support the following encodings in Xaml Documents: UTF-8 and UTF-16. (These encodings are defined in section 3.9 of The Unicode Standard [Unicode], in definitions D92 and D91 respectively.)

```
Let xamlRoot = Invoke "Object Node Creation from an XML:element"
                            (xmlObjectElement ::= xmlDocument[document element],
                            parentPreservesXmlSpace ::= False)
```

Comments and processing instructions in *xmlDocument*[children] MUST be ignored.

Let result be a Document Information Item initialized as follows:

```
result[document object] = xamlRoot
```

This rule MUST return *result*.

8.6.2 Object Node Creation from an XML:element

This rule has the following input:

Name	Type	Purpose (non-normative)
xmlObjectElement	XML:element	An XML element representing an object.
parentPreservesXmlSpace	Boolean	Indicates whether xml:space="preserve" is in effect for this node's parent.

The output of this rule is a Xaml Information Set Object Node Information Item.

Let objectType be a XamlType Information Item determined as follows:

- If schema(*xmlObjectElement*)[types] contains a type 't' where t[name] matches *xmlObjectElement*[local name], let objectType be that t.
- Otherwise, if schema(*xmlObjectElement*)[types] contains a type 't' where t[name] matches the concatenation of *xmlObjectElement*[local name] and 'Extension' and t[types assignable to] contains the x:MarkupExtension type (section 7.2.22), let objectType be that t.
- Otherwise, it is an unknown element type error.

Let preserveChildXmlSpace be a Boolean value determined as follows:

- If *xmlObjectElement*[attributes] contains an XML:attribute corresponding to the standard xml:space attribute, let preserveChildXmlSpace be True if the attribute's [normalized value] is "preserve", and let it be False otherwise.
- Otherwise, let preserveChildXmlSpace be *parentPreservesXmlSpace*.

Let childXmlNodes be *xmlObjectElement*[children].

If objectType is either of the intrinsic types x:Code (section 7.2.23) or x:XData (section 7.2.24), then:

- Let literalResult be an Object Node Information Item initialized as follows:

```
literalResult[type] = objectType
literalResult[member nodes] is set to a single text node, whose value
is a string representation of xmlObjectElement[children]
literalResult[parent member] is not determined by the rules in this section
--it is set by the rule from which this rule was invoked, or in the case of
the root element, it is Null.
literalResult[is retrieved] is False
literalResult[xml namespace mappings] is generated as follows:
    Invoke "Xml Namespace Mapping Conversion"
         (xmlNamespaces ::= xmlObjectElement[in-scope namespaces])
```

- The output of this rule is literalResult.

Otherwise, proceed with the remaining steps in this rule.

The nodes in childXmlNodes are processed in phases to produce a sequence of information items. The following list shows a summary of the phases, which are described in detail below.

1. Conversion: childXmlNodes is converted into a sequence of items, where each item is either an object node, a member node, or a text node.

2. Whitespace removal: text nodes between member nodes, or before the first member node, or after the last member node, are stripped out.

3. Content wrapping: text or object nodes are wrapped in implicit member nodes; the sequence now contains only member nodes.

Conversion

Conversion builds an intermediate result, convertedChildNodes, which is an ordered sequence that may contain Object Node Information Items, Member Node Information Items, or Text Node Information Items. It is generated by applying the first of the following steps that matches to each xmlChild in childXmlNodes in document order.

- If xmlChild is an XML:element:
  - If xmlChild[local name] is a XamlName:

```
Let convertedObject = Invoke "Object Node Creation from an XML:element"
        (xmlObjectElement ::= xmlChild,
        parentPreservesXmlSpace ::= preserveChildXmlSpace)
```

    then append convertedObject to convertedChildNodes.

  - Otherwise, if xmlChild[local name] is a DottedXamlName:

```
Let convertedMember = Invoke "Member Node Creation from an XML:element"
        (xmlMemberElement ::= xmlChild,        containingType ::= xmlObjectElement,
        parentPreservesXmlSpace ::= preserveChildXmlSpace)
```

    then append convertedMember to convertedChildNodes.

  - Otherwise, it is an "Invalid element name syntax" error.
- If xmlChild is an XML:character:
  - If convertedChildNodes ends in a Text Node Information Item, append the Unicode character identified by xmlChild[character code] to that node's [text].
  - Otherwise, append to convertedChildNodes a new Text Node Information Item with a [text] value containing the Unicode character identified by xmlChild[character code]. (The [parent member] will be set later.)
- An XML node of any other type is ignored, and will not cause an item to be added to convertedChildNodes.

If convertedChildNodes contains any Object Node Information Item with a [type] of the intrinsic x:Code (section 7.2.23), remove these from convertedChildNodes. If convertedChildNodes contains an Member Node Information Items whose [values] contain any Object Node Information Items with a [type] of the intrinsic x:Code (section 7.2.23), remove these from that [values].

Let codeItems be the Object Node Information Items removed by the steps in the previous paragraph. If codeItems is not empty, add a new Member Node Information Item called directiveChildren initialized as follows:

directiveChildren[member] is the intrinsic x:DirectiveChildren (section 7.3.12)XamlMember Information Item.

directiveChildren[values] contains the Object Node Information Items in codeItems, with the [parent member] of each item set to directiveChildren.

directiveChildren[parent object] is set later on in this section.

directiveChildren[xml namespace mappings] is generated as follows:

```
Invoke "Xml Namespace Mapping Conversion"
        (xmlNamespaces ::= xmlObjectElement[in-scope namespaces])
```

Whitespace Removal

Whitespace removal takes convertedChildNodes and strips out whitespace between member nodes to form a new intermediate result, strippedChildNodes. For the purposes of this section, 'whitespace characters' are those listed in section 8.5.2, "Collapsible Whitespace Characters". Removal proceeds as follows:

For each node convertedChild in convertedChildNodes:

- If convertedChild is a Text Node Information Item, and if its [text] contains only whitespace, then ignore the node if any of the following is true:
  - convertedChild is the first item in convertedChildNodes, and is followed immediately by a member node.
  - convertedChild immediately follows a member node and is followed immediately by a member node.
  - convertedChild immediately follows a member node and is the last item in convertedChildNodes, and either strippedChildNodes already contains one or more text nodes, or convertedChildNodes contains a member node whose [member] is objectType[content property].
- Otherwise, append convertedChild to strippedChildNodes.

Content Wrapping

Content wrapping proceeds as follows. Define contentMember and contentMemberType with the first of the following to match:

- If objectType[content property] is not Null, let contentMember be objectType[content property], and let contentMemberType be contentMember[value type].
- Otherwise, let contentMember be the intrinsic x:Items (7.3.1), and let contentMemberType be objectType.

Let attributeMembers be the set of Member Node Information Items generated by applying the following invocation for each xmlAttribute in xmlObjectElement[attributes]:

```
Invoke "Member Node Creation from an XML:attribute"
    (xmlAttribute ::= xmlAttribute,
    objectType ::= objectType,
    namespacesInScope ::= xmlObjectElement[in-scope namespaces])
```

Let members be a set of Member Node Information Items. If all of the following are True:

- strippedChildNodes contains exactly one Text Node Information Item and strippedChildNodes either contains no other items, or contains only a Member Node Information Items whose [member] is the intrinsic x:DirectiveChildren (section 7.3.12).
- attributeMembers is either empty, or contains only Member Node Information Items whose [member] is either the x:Key Directive (section 7.3.5) or the x:Uid Directive (section 7.3.6).
- Either contentMember[text syntax] is not Null or objectType[text syntax] is not Null.

then let members contain just a single Member Node Information Item initTextMember initialized as follows:

initTextMember[member] is the intrinsic x:Initialization (section 7.3.3) member.

initTextMember[parent object] is set later in this rule.

initTextMember[values] contains the single text node in strippedChildNodes.

Otherwise, let members be the union of the following three sets:

- attributeMembers.
- The set of Member Node Information Items generated by applying the following invocation for each consecutive sequence npChildren of non-member-node items in strippedChildNodes:

```
Invoke "Member Node Creation from Content"
    (containingMember ::= contentMember,
    memberType ::= contentMemberType,
    childNodes ::= npChildren,
    preserveXmlSpace ::= preserveChildXmlSpace,
    namespacesInScope ::= xmlObjectElement[in-scope namespaces])
```

- The set of items in strippedChildNodes that are Member Node Information Items.

Let result be an Object Node Information Item initialized as follows:

result[type] = objectType.

result[member nodes] is set to members, with [parent object] of each Member Node Information Item set to result.

result[parent member] is not determined by the rules in this section - it is set by the rule from which this rule was invoked, or in the case of the root element, it is Null.

result[is retrieved] is False.

result[xml namespace mappings] is generated as follows:

```
Invoke "Xml Namespace Mapping Conversion"
    (xmlNamespaces ::= xmlObjectElement[in-scope namespaces])
```

This rule MUST return *result*.

8.6.2.1 Notes (Non-Normative)

The content wrapping process can end up producing more than one content member. For example, consider the following XML:

```
<MyObject>
  Some content
  <MyObject.Prop>BarValue</MyObject.Prop>
  More content
</MyObject>
```

If <MyObject> represents an object node, its [member nodes] will contain two content nodes, one representing each piece of text. The well-formedness rules in section 6 require that Xaml processors identify this as an error--a Xaml Information Set is not well formed if an object node contains two member nodes with the same XamlMember Information Item. (See 6.2.1.3.) However, for some applications (e.g. Xaml editors) it may be useful to carry on document processing even when a document is known not to be valid. The example XML shown above is conceptually equivalent to this:

```
<MyObject>
  <MyObject.Content>
    Some content
  </MyObject.Content>
  <MyObject.Prop>BarValue</MyObject.Prop>
  <MyObject.Content>
    More content
  </MyObject.Content>
</MyObject >
```

Where 'Content' is MyObject's content member. This makes it more obvious why this is not considered well-formed Xaml. The error is the same in both cases - the same member appearing twice in one element. The only difference in the first example is that the content member has not been spelled out as a member element.

8.6.3 Member Node Creation from an XML:attribute

This rule has the following inputs:

Name	Type	Purpose (non-normative)
xmlAttribute	XML:attribute	An XML attribute present on an element representing an object.
objectType	XamlType Information Item	The type of object that the attribute's containing element represents.

Name	Type	Purpose (non-normative)
namespacesInScope	Set of XML:namespaces	Namespaces in scope for the element that contains this attribute.

The output of this rule is a Xaml Information SetMember Node Information Item.

Let member be a XamlMember Information Item determined by the following:

- If *xmlAttribute*[local name] is a XamlName:
  - If schema(*xmlAttribute*)[types] contains *objectType* schema(*xmlAttribute*)[compatible with schemas] contains a schema whose [types] contains *objectType* or a schema is available whose [types] contains *objectType* and whose [compatible with schemas] contains schema(*xmlAttribute*):
    - Define memberInfo as follows:

```
Let memberInfo = Invoke "Member Lookup"
      (definingType ::= objectType,
      memberName ::= xmlAttribute[local name])
```

    - If memberInfo is not Null, let member be memberInfo.
  - If the previous step does not determine a value for member, if there exists a XamlMember Information Item in schema(*xmlAttribute*)[directives] for which the [name] property matches *xmlAttribute*[local name], let member be that XamlMember Information Item.
  - If neither of the previous steps determines a value for member, it is an unknown member error.
- If *xmlAttribute*[local name] is a DottedXamlName:
  - Let typeName be the DottedXamlName's typename.
  - Let memberName be the DottedXamlName's membername.
  - Let definingType be a XamlType Information Item determined as follows:
    - If schema(*xmlAttribute*)[types] contains *objectType* and *objectType*[types assignable to] contains a XamlType Information Item where [name] matches typeName, let definingType be *objectType*.
    - Otherwise, if schema(*xmlAttribute*)[types] contains a XamlType Information Item xt where xt[name] matches typeName, let definingType be xt.
    - Otherwise, it is an unknown type error.
  - Define memberInfo as follows:

```
Let memberInfo = Invoke "Member Lookup"
    (definingType ::= definingType,
    memberName ::= memberName)
```

- If memberInfo is not Null, let member be memberInfo.
- Otherwise it is an unknown member error.

- Otherwise, it is an invalid attribute syntax error.

If member[allowed location] is None, it is an "unknown member" error.

Let attributeText be the string *xmlAttribute*[normalized value].

The member will contain a single value, attributeValue, which is calculated as follows:

```
Let attributeValue = Invoke "Value Creation from Attribute Text"
        (valueText ::= attributeText,
        attributeMemberType ::= member[value type],
        namespacesInScope ::= namespacesInScope)
```

If attributeValue is an Object Node Information Item, its [parent member] is set to result. If attributeValue is a Text Node Information Item, its [parent member] is set to result.

Let result be a Member Node Information Item initialized as follows:

```
result[member] = member
result[parent object] is not set by this rule - it is set by the invoker of this rule
result[values] = a collection containing a single item, attributeValue
```

This rule MUST return *result*.

8.6.3.1 Notes (Non-Normative)

The rules in this section consider attributes that match the XamlName production, and that are not qualified with a namespace prefix to be equivalent to attributes explicitly placed in the same namespace as their containing element. For example, this:

```
<q:MyObject Prop="42" />
```

Is equivalent to this:

```
<q:MyObject q:Prop="42" />
```

In particular, note that this means an unqualified attribute is not necessarily considered to be in the default namespace. That will only be the case if the element happens to be in the default namespace in scope. So if d: is a prefix for the namespace that is also the default namespace in scope, the following are all equivalent:

```
<AnotherObject AnotherProp="99" />
<d:AnotherObject AnotherProp="99" />
<AnotherObject d:AnotherProp="99" />
<d:AnotherObject d:AnotherProp="99" />
```

Attributes representing attached properties are handled differently. If they are not qualified with a namespace prefix, they are considered to be equivalent to attributes explicitly placed in the default namespace.

8.6.4 Value Creation from Attribute Text

This rule has the following inputs:

Name	Type	Purpose (non-normative)
valueText	Text String	The text value to represent as a value. (Either an attribute value, or a string nested inside a markup extension attribute value.)
attributeMemberType	XamlType Information Item	Type of value this text represents.
namespacesInScope	Set of XML:namespaces	The set of namespaces in scope on the element in which this text value appears.

The output of this rule will either be an Object Node Information Item or a Text Node Information Item.

Let result be defined as follows:

- If *valueText* begins with '{' and does not begin with '{}', the value is an Object Node Information Item determined as follows:

```
Let result = Invoke "Object Node Creation from a Markup Extension in an Attribute"
                (attributeText ::= valueText,
                namespacesInScope ::= namespacesInScope)
```

- Otherwise, let unescapedValue be a Text String defined as follows:
  - if *valueText* begins with '{}', unescapedValue is *valueText* with the leading '{}' removed.
  - Otherwise, unescapedValue is *valueText*.
- Let result be a new Text Node Information Item whose [text] is unescapedValue and whose [parent member] is set by the invoker of this rule.

This rule MUST return *result*.

8.6.5 Member Node Creation from an XML:element

This rule has the following inputs:

Name	Type	Purpose (non-normative)
xmlMemberElement	XML:element	An XML element representing a member.
containingType	XamlType Information Item	The type of the object node to which this member will belong.
parentPreservesXmlSpace	Boolean	Indicates whether xml:space="preserve" is in effect for this node's parent.

The output of this rule is a member node. This does not set the [parent object] property--the invoker of this rule must provide the value for that property.

If *xmlMemberElement*[attributes] is not empty, it may only contain a single attribute, whose [local name] MUST be "Uid", and whose [namespace name] MUST be the [target namespace] of the x: Schema ("http://schemas.microsoft.com/winfx/2006/xaml"). If *xmlMemberElement*[attributes] contains anything other than this, it is a "member elements cannot contain attributes" error. Xaml processors MAY choose to continue processing by ignoring any attributes. (Note that in the case where the x:Uid (section 7.3.6) attribute is present, it is ignored--no Xaml Information Set elements are created to represent the attribute. The only difference between this and any other attribute is that the x:Uid's presence is not deemed to be an error.)

This rule is only invoked when *xmlMemberElement*[local name] is a DottedXamlName. Let elementTypeName and elementMemberName be the typename and membername respectively of this DottedXamlName.

Let ownerType be a XamlType Information Item determined as follows:

- If schema(*xmlMemberElement*)[types] contains a type 't' where t[name] matches elementTypeName, then:
  - If *containingType*[types assignable to] contains t, let ownerType be *containingType*.
  - Otherwise, let ownerType be t.
- Otherwise, it is an unknown element type error.

Let resolvedMember be a XamlMember Information Item determined as follows:

- Define memberInfo as follows:

```
Let memberInfo = Invoke "Member Lookup"
        (definingType ::= ownerType,
        memberName ::= elementMemberName)
```

- If memberInfo is not Null, let resolvedMember be memberInfo.
- Otherwise, it is a "member not found" error.

If memberInfo [allowed location] is not Any, it is a "member not found" error.

Let convertedChildNodes be an ordered sequence that may contain Object Node Information Items or Text Node Information Items. (Note: The following part is similar to the conversion process in the Object Node Creation from an XML:element (section 8.6.2) rule. The difference is that here, member elements are not permitted as children, since these elements are already inside a member element.) It is generated by applying the first of the following steps that matches to each xmlChild in childXmlNodes in document order.

- If xmlChild is an XML:element:
  - If xmlChild[local name] is a XamlName:

```
Let convertedObject = Invoke "Object Node Creation from an XML:element"
          (xmlObjectElement ::= xmlChild,
          parentPreservesXmlSpace ::= parentPreservesXmlSpace)
```

then append convertedObject to convertedChildNodes.

- Otherwise, if xmlChild[local name] is a DottedXamlName, it is a "Member elements may not be nested directly inside of another member element" error.
- Otherwise, it is an "Invalid element name syntax" error.

- If xmlChild is an XML:character:
  - If convertedChildNodes ends in a Text Node Information Item, append the Unicode character identified by xmlChild[character code] to that node's [text].
  - Otherwise, append to convertedChildNodes a new Text Node Information Item with a [text] value containing the Unicode character identified by xmlChild[character code]. (The [parent member] will be set later.
- An XML node of any other type is ignored, and will not cause an item to be added to convertedChildNodes.

```
Let result = Invoke "Member Node Creation from Content"
                (containingMember ::= resolvedMember,
                memberType ::= resolvedMember[value type],
                childNodes ::= convertedChildNodes,
                preserveXmlSpace ::= parentPreservesXmlSpace,
                namespacesInScope ::= xmlObjectElement[in-scope namespaces])
```

This rule MUST return *result*.

8.6.6 Member Node Creation from Content

This rule has the following inputs:

Name	Type	Purpose (non-normative)
containingMember	XamlMember Information Item	The kind of member to create.
memberType	XamlType Information Item	The type of the member. (This is not always the same as *containingMember*[value type]. If this rule is invoked for the implicit content of a list-like object node, *containingMember* will be the intrinsic x:Items (Constraints) member, while *memberType* will be the containing object's type.)
childNodes	Ordered Collection of items, where each item is either an Object Node Information Item or a Text Node Information Item	The text and object nodes to be processed and wrapped.
preserveXmlSpace	Boolean	Indicates whether XML space preservation is on for the elements to be processed.
namespacesInScope	Set of XML:namespaces	Namespaces in scope for the element that contains this content.

The output of this rule is a Member Node Information Item.

For the purposes of this section, 'whitespace characters' are those listed in section 8.5.2, "Collapsible Whitespace Characters". If *preserveXmlSpace* is False, apply the following steps:

- For each Text Node Information Item in *childNodes*, any linefeed character (LF--Unicode Code Point 000A) both preceded and followed by a character in the ranges defined in section 8.5.3, 'Linefeed Collapsing Characters' is removed.
- For each Text Node Information Item in *childNodes*, replace any sequence of whitespace characters with a single space.
- If *childNodes* contains at least one Text Node Information Item, trim any leading whitespace from the first Text Node Information Item; if this leaves the node's [text] empty, remove the node from *childNodes*.
- If *childNodes* contains at least one Text Node Information Item, trim any trailing whitespace from the last text node; if this leaves the node's [text] empty, remove the node from *childNodes*.
- For each object node 'obn' in *childNodes*, if obn[type][trim surrounding whitespace] is True, apply the following rules:
  - If *childNodes* contains a text node as the item before obn, trim any trailing whitespace from that text node; if this leaves the text node empty, remove it from *childNodes*.
  - If *childNodes* contains a text node as the item after obn, trim any leading whitespace from that text node; if this leaves the text node empty, remove it from *childNodes*.

If *memberType* [is whitespace significant collection] is False, then for each text node 'tn' in *childNodes*, trim leading and trailing whitespace; if this leaves the text node empty, remove it from *childNodes*.

Let outputValues be an ordered collection of information items, where items are either Object Node Information Items, or Text Node Information Items, determined as follows:

- If either *memberType*[is list] or *memberType*[is dictionary] are True, and *childNodes* does not contain a single Object Node Information Item obj where [type][types assignable to] contains *memberType*, then:

```
let retrievedContentMember be a new Member Node Information Item where
        retrievedContentMember[member] is the intrinsic x:Items XamlMember Information
Item
        retrievedContentMember[values] is childNodes
        retrievedContentMember[parent object] the retrievedMemberValue defined next
let retrievedValue be a new Object Node Information Item where
        retrievedValue[type] is memberType
        retrievedValue[member nodes] contains just retrievedContentMember
        retrievedValue[parent member] is set below
        retrievedValue[is retrieved] is True
        retrievedValue[xml namespace mappings] is is generated as follows:
                Invoke "Xml Namespace Mapping Conversion"
                        (xmlNamespaces ::= namespacesInScope)
```

  and let outputValues contain just retrievedValue.

- Otherwise, let outputValues be *childNodes*.

Let result be a new Member Node Information Item prop, initialized as follows:

```
result [member] is containingMember
result [parent object] is supplied by the invoker of this rule
result [values] outputValues, where for each text node tn in values set tn[parent member]
    to prop; for each object node on in value set on[parent member] to prop
```

This rule MUST return *result*.

8.6.7 Object Node Creation from a Markup Extension in an Attribute

This rule has the following inputs:

Name	Type	Purpose (non-normative)
attributeText	Text String	The text value of the attribute to be handled as a markup extension
containingElementSchema	Schema Information Item	The schema of the element that contains this attribute
namespacesInScope	Set of XML:namespaces	Namespaces in scope for the element that contains this attribute

The output of this rule is an object node.

Handling of Markup Extension attribute values is performed in two steps. First, the value MUST be parsed. Then the resulting parse tree MUST be converted into Xaml Information Set nodes.

8.6.7.1 Markup Extension Parsing

The *attributeText* input MUST match the MarkupExtension production in the following grammar (using the syntax defined in [RFC4234]):

```
MarkupExtension = "{" TYPENAME 0*1Arguments "}"
Arguments       = (NamedArgs / (PositionalArgs 0*1("," NamedArgs))
NamedArgs       = NamedArg *("," NamedArg)
NamedArg        = MEMBERNAME "=" STRING
PositionalArgs  = NamedArg / (STRING 0*1( "," PositionArgs))
```

TYPENAME, MEMBERNAME and STRING above are special - they are not defined in the grammar because they are terminals as far as this ABNF grammar is concerned. This is because Markup Extensions have slightly unusual tokenization rules. With normal ABNF, the input to the grammar is a sequence of characters. But with Markup Extensions, a more complex tokenization process is performed, which presents *attributeText* as a sequence consisting of the tokens shown in Table 91.

Table 91: Markup Extension tokens

Token Name	Description
"{"	Opening brace
"}"	Closing brace

Token Name	Description
"="	Equals sign
","	Comma.
TYPENAME	String representing the name of a type. Follows a "{" token, space delimited.
MEMBERNAME	String representing the name of a type. Precedes an "=" token. Delimited by any of "{" "}" "," or "=" tokens.
STRING	String representing the value of a member. Delimited by any of {},= tokens.

The tokenizer treats as 'whitespace' the characters identified in section 8.5.2, "Collapsible Whitespace Characters" (i.e., the Unicode code points 0009, 000A, and 0020).

The tokenizer consumes characters in *attributeText* and generates tokens as follows:

The tokenizer starts by consuming the initial "{" (Unicode code point 007B) and generating a "{" token.

The tokenizer consumes any whitespace characters that immediately follow the "{" character, and does not produce a token to represent these whitespace characters.

Next, the tokenizer generates a TYPENAME token to represent the sequence of non-whitespace characters that immediately follows the "{" character. The tokenizer consumes all the characters up to, but not including the first character that is either whitespace or a "}" (Unicode code point 007D) character, and those consumed characters are the TYPENAME's value.

Next, the tokenizer consumes all characters up to but not including the first non-whitespace character, and does not produce a token to represent these whitespace characters.

The tokenizer proceeds by executing the following rules until all the characters in *attributeText* have been consumed:

If the next character is a "}" (Unicode code point 007D), consume the character and generate a "}" token.

If the next character is an "=" (Unicode code point 003D), consume the character and generate an "=" token.

If the next character is a "," (Unicode code point 002C), consume the character and generate a "," token.

If the next character is not one of "{}=," then it is the start of either a MEMBERNAME or a STRING. To determine which kind of token to produce, the tokenizer MUST first read a text value (see below), and then examine (but not consume) the character immediately following this text value. If the character is "=", the token is a MEMBERNAME. Otherwise, it is a STRING. If the token is a MEMBERNAME, any leading and trailing whitespace is removed from the text value.

The text value of either a MEMBERNAME or STRING is read as follows. Leading whitespace characters are consumed without being represented in the generated token. If the first non-whitespace character is a quote (either Unicode code point 0022, Quotation Mark, or 0027, Apostrophe), the tokenizer proceeds as follows:

- The first quote is consumed and is not represented in the token's value.

- The text value becomes the characters up to but not including the next matching quote (i.e. a character of the same code point as the opening quote) that is not preceded by a "\" character. All these characters and also the closing quote are consumed. Any "\" characters in the resulting text value are removed.
- Whitespace characters following the closing quote are consumed, and are not represented in the token.

But if the first non-whitespace character of the text value is not a quote, then instead, the text value is formed as follows. Initialize the running brace total to 0, then repeat the following rules until they determine that the process is complete:

- If the next character is a "\" (Unicode code point 005C), consume that "\" without adding it to the text value, then consume the following character and append that to the value.
- If the next character is a "{", add one to the running brace total, and consume the character, appending it to the value.
- If the next character is a "}", then the behavior depends on the running brace total:
  - If the running brace total is zero, do not consume the "}", and do not append any more to the text value; the text value reading process is now complete.
  - If the running brace total is non-zero, consume the "}", append it to the value, subtract one from the running brace total, and continue the text value reading process.
- If the next character is a "," or an "=", do not consume that character; the text reading process is now complete.
- Otherwise, consume the character and append it to the value.

The resulting sequence of characters then has leading and trailing whitespace characters removed. The result is the value text.

8.6.7.2 Converting Parsed MarkupExtension to Xaml Information Set Nodes

The parsing procedure described in the previous section will result in a parse tree with a MarkupExtension production at its root. This is converted into Xaml Information Set nodes as follows.

Let extensionType be a XamlType Information Item determined as follows:

- Let typeName be the text value of the TYPENAME production of the root MarkupExtension production.
- If typeName is not a QName (as defined in section 4 of [XML Namespaces]), it is a 'bad type extension name' error.
- Let extensionSchema be a Schema Information Item:
  - If typeName is an UnprefixedName (as defined in section 4 of [XML Namespaces]), let extensionSchema be *containingElementSchema*.
  - If typeName is a PrefixedName (as defined in section 4 of [XML Namespaces]); perform the following steps:
    - Let typeNamespaceUri be the [uri] of the XML namespace mapping in *namespacesInScope* whose [prefix] matches the typeName's Prefix (as defined in section 4 of [XML

Namespaces]); if no such XML namespace mapping exists, it is a 'unrecognized namespace prefix' error.

    - Let extensionSchema be the schema whose [target namespace] matches typeNamespaceUri. If no such schema is available, this is an "Unknown namespace" error.

- Let typeLocalName be the typeName's LocalPart (as defined in section 4 of [XML Namespaces]).
- Let extensionType be the XamlType Information Item in extensionSchema[types] whose [name] matches the concatenation of typeLocalName and the string "Extension"; if no such XamlType Information Item exists or if extensionType[types assignable to] does not include the intrinsic x:MarkupExtension (section 7.2.22), let extensionType be the XamlType Information Item in extensionSchema[types] whose [name] matches typeLocalName; if extensionType[types assignable to] does not include the intrinsic x:MarkupExtension (section 7.2.22), or if no such XamlType Information Item exists, it is a 'unknown markup extension' error.

Let namedArgs be the set of NamedArg productions in the parse tree.

Let namedMembers be a set of Member Node Information Items generated as follows. For each namedArg in namedArgs perform the following steps:

- Let memberName be the MEMBERNAME production in namedArg.
- If memberName is not a QName (as defined in section 4 of [XML Namespaces]), it is a 'bad member name' error; if it is a QName, let memberLocalName be the LocalPart of the QName.
- Let memberSchema be defined as follows:
    - If memberName is a UnprefixedName (as defined in section 4 of [XML Namespaces]), let memberSchema be extensionSchema.
    - Otherwise, memberName is a PrefixedName (as defined in section 4 of [XML Namespaces]); perform the following steps:
        - Let memberNamespaceUri be the [uri] of the XML namespace mapping in *namespacesInScope* whose [prefix] matches the memberName's Prefix (as defined in section 4 of [XML Namespaces]); if no such XML namespace mapping exists, it is an 'unrecognized namespace prefix' error.
        - Let memberSchema be the schema whose [target namespace] matches memberNamespaceUri. If no such schema is available, this is an "Unknown namespace" error.
- Let member be the Member Node Information Item this named argument represents, determined as follows:
    - If memberLocalName is a XamlName:
        - If memberSchema[types] does not contain extensionType, then it is a 'markup extension named members MUST either be in the same schema as the extension, or be attached members' error.
        - Define memberInfo as follows:

```
Let memberInfo = Invoke "Member Lookup"
            (definingType ::= extensionType,
            memberName ::= memberLocalName)
```

- If memberInfo is not Null, let member be memberInfo.
- Otherwise, it is an unknown member error.

- Otherwise, If memberLocalName is a DottedXamlName:
  - Let typeName be the DottedXamlName's typename.
  - Let memberName be the DottedXamlName's membername.
  - Let definingType be a XamlType Information Item determined as follows:
    - If memberSchema[types] contains a XamlType Information Item where [name] matches typeName, let definingType be that type.
    - Otherwise, it is an unknown type error.
  - Define memberInfo as follows:

```
Let memberInfo = Invoke "Member Lookup"
             (definingType ::= definingType,
             memberName ::= memberName)
```

  - If memberInfo is not Null, let member be memberInfo.
  - Otherwise it is an unknown member error.
- Let memberValueText be the STRING production in namedArg.
- Define memberValue as follows:

```
Let memberValue = Invoke "Value Creation from Attribute Text"
            (valueText ::= memberValueText,
            attributeMemberType ::= member[value type],
            namespacesInScope ::= namespacesInScope)
```

- Defined namedMember node as a new Member Node Information Item initialized as follows:

```
namedMember[member]is member
namedMember[values] contains just memberValue
namedMember[parent object] is set later in this rule
```

- Add resulting node to namedMembers.

Let positionalArgs be the set of STRING productions that are descendants of the PositionalArgs production in the Arguments production in MarkupExtension production, ordered by increasing depth in the parse tree.

Let positionalArgValues be an ordered sequence of Object Node Information Items and Text Node Information Items generated as follows. Let constructorInfo be the Constructor Information Item for the type whose [arguments] contains as many items as there are in positionalArgs. For each positionalArg in positionalArgs perform the following steps:

- Let argumentType be the Nth item in constructorInfo[arguments] where N is the characterised thus: the current positionalArg is the Nth item in positionalArgs.
- Calculate argValue, the value of the argument, thus:

```
Let argValue = Invoke "Value Creation from Attribute Text"
        (valueText ::= argValueText,
        attributeMemberType ::= argumentType,
        namespacesInScope ::= namespacesInScope)
```

- Append argValue to positionalArgValues.

If positionalArgValues is non-empty, let positionalArgsMember be a new member node where:

```
positionalArgsMember[member] is the intrinsic x:PositionalParameters
positionalArgsMember[values] is positionalArgValues, where each Object Node Information
     Item has its [parent member] set to positionalArgsMember and each Text Node
     Information Item has its [parent member] set to positionalArgsMember
positionalArgsMember[parent object] is set below.
```

Otherwise, positionalArgValues is empty, in which case let positionalArgsMember be Null.

Let allArgs be a set of Member Node Information Items that contains all the items in namedMembers, and, if positionalArgsMember is non-Null, it also contains positionalArgsMember.

Let result be a new Object node:

```
result[type] is extensionType
result[member nodes] is allArgs, where the [parent object] property of each item
    is set to result
result[parent member] is
result[is retrieved] is False
result[xml namespace mappings] is namespacesInScope
```

This rule MUST return result.

8.6.8 Member Lookup

This rule has the following inputs:

Name	Type	Purpose (non-normative)
definingType	XamlType Information Item	The type on which to look up the member
memberName	Text String	The name of the member to find

The rule's output is a XamlMember Information Item.

If *definingType*[members] contains a member prop where prop[name] matches *memberName*, this rule MUST return prop. Otherwise, this rule MUST return Null.

8.6.9 Xml Namespace Mapping Conversion

This rule has the following inputs:

Name	Type	Purpose (non-normative)
xmlNamespaces	Set of XML:namespace	A set of XML namespace mappings from an XML Infoset

The rule's output is a set of XML namespace mappings, generated as follows:

For each XML:namespace xmlNsMapping in *xmlNamespaces* create an XML namespace mapping, setting [uri] to xmlNsMapping[namespace name] and [prefix] to xmlNsMapping[prefix]. This rule MUST return all of the XML namespace mappings thus generated.

9 References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[ECMA-376] ECMA International, "Office Open XML File Formats", 1st edition, ECMA-376, December 2006, http://www.ecma-international.org/publications/standards/Ecma-376.htm

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

[RFC3986] Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005, http://www.ietf.org/rfc/rfc3986.txt

[RFC4234] Crocker, D., Ed., and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005, http://www.ietf.org/rfc/rfc4234.txt

[UNICODE5.0.0/2007] The Unicode Consortium, "Unicode 5.0.0", 2007 http://www.unicode.org/versions/Unicode5.0.0/

[XML] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", W3C Recommendation, August 2006, http://www.w3.org/TR/2006/REC-xml-20060816/

[XML-INFOSET] Cowan, John, and Tobin, Richard, "XML Information Set (Second Edition)", W3C Recommendation, February 2004, http://www.w3.org/TR/2004/REC-xml-infoset-20040204

[XML-Namespaces] Bray, T., Hollander, D., and Layman, A., "Namespaces in XML", W3C Recommendation, January 1999, http://www.w3.org/TR/1999/REC-xml-names-19990114/

[XMLSCHEMA2/2] Biron, P.V., and Malhotra, A., Eds., "XML Schema Part 2: Datatypes Second Edition", W3C Recommendation, October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

10 Microsoft .NET Framework Behavior

The information in this specification is applicable to the following versions of the Microsoft product:

- .NET Framework 4.0

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies .NET Framework behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that .NET Framework does not follow the prescription.

11 Index