XBRL Formula Requirements

Public Working DRAFT of Tuesday, 20 April 2004

This file:

Formula-Req-PWD-2004-04-20.doc

Updates the public working draft originally posted at:

http://www.xbrl.org/tr/2002/XBRL-Rule-Base-Req-WD-2002-11-16.doc

Editors

Name	Contact	Affiliation
Walter Hamscher[1]	walter@hamscher.com	Standard Advantage

Contributors

Geoff Shuetrim	gshuetrim@kpmg.com.au	KPMG LLP
David vun Kannon	dvunkannon@kpmg.com	KPMG LLP

Status of this document

This is a public working draft whose circulation is unrestricted. It may change and is not appropriate to reference from public documents. Comments should be directed to the editor and/or contributors by e-mail. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Table of contents

[1] Walter Hamscher is a consultant to PricewaterhouseCoopers.

1 Motivation (non-normative)

To effectively exchange business reporting information between a producing and a consuming application often requires the applications to perform validation on data types and values, apply consistency checks, test data quality, augment the data with calculated values and possibly corrections, and to provide feedback to the producing application that indicates the nature and severity of problems encountered. A producing application may also add calculated values and perform tests on its own outputs before sending results to consuming applications. Applications based on the exchange of spreadsheets that contain both data and formulas are common but present maintenance problems; here the intent is explicitly to separate the representation and location of formulas and validation criteria from a given instance.

These different functions are characteristic of many XBRL-enabled applications. That is because it is a goal of XBRL to allow applications to consume XBRL formatted business information no matter what its original source. Data entry validation (e.g., prohibiting dates such as February 31, 2001 from entering the system) is a familiar example of using a formula to flag errors, but the breadth and scope of formulas is much broader than this. Good information management practice generally recognizes that validations should be tested, adhered to, and corrections made as far "upstream" in an information supply chain as possible – recognizing, of course, that the same tests may be applied repeatedly as data makes its way from its many possible origins, to its many ultimate destinations.

Any general programming language could be used to perform computations on business data. However, there is regularity in business reporting information: regularity that is encoded in XBRL in instance constructs such as "facts," "periods," "entities," and in taxonomies as items, definitions, and other relationships. Furthermore, applications that consume XBRL should be able to "publish the formulas" that govern documents containing XBRL data, so that producing applications can test and apply those formulas and thereby reduce delay, rework and retransmissions, and smooth and accelerate the flow of business reporting information.

Rule languages—in which rules are expressed as patterns to be matched and actions to be taken when the data matches the pattern—have a rich tradition and are nearly as old as Computer Science itself ([Newell, 1962]), enjoyed a heyday in the 1980's for the programming of "expert systems", and today rule languages live on in languages such as Prolog, and in commercial products such as the Blaze Rule Engine [HNC] and JRules [ILOG]. Rule languages tend to be fairly compact and concise and resemble a database of logical sentences ("all people are mortal" "if X is a mountain, then the elevation of X must be positive"). They declare the rules to be obeyed, and an interpreter (or compiler) is responsible for efficiently executing the rules (matching patterns and executing actions) correctly when presented with incoming data.

Even if we set aside the goal of synthesizing information and are content merely to detect violation of constraints, XML Schema [XSDL] itself is still not sufficient to this task because it allows the validation of individual data elements ("Revenues are a 12-digit nonnegative number") and structural relationships (e.g., "an invoice must contain a header, a list of items, and an amount") but does not express constraints *between* elements, also known as co-constraints ("if more than 10 dependents are claimed then Schedule K must be completed"). Besides, the nature of rules is that they generally compute a whole series of results, some of which may be considered fatal errors, others as warnings, others as merely informational; an XML Schema validator mainly detects fatal errors. Other general programming languages, including XML Schema Transformation [XSLT], could be pressed into service since they have the requisite pattern-action structure, but these neither take any account nor take any advantage of the constrained nature of XBRL instances, taxonomies, and so on.

XBRL-specific formulas are well motivated. The language would be, like XBRL itself, suitable for publishing and transporting between applications; it would exploit the XBRL language itself, and provide a concise and well constrained way of expressing common rules such as "Net Receivables cannot be negative," "Net revenues are the difference between Gross revenues and Gross expenses," "Unless income for the previous quarter was zero, income growth for

the quarter is the ratio of the change in income between the current and previous quarter, divided by the income of the previous quarter," and so on. The earliest discussions of XBRL acknowledged the need to express not only specific numeric facts ("1988 Revenue for TLA Inc. was $40m"), but also relationships among those facts ("TLA Inc. revenue consists of TBD Inc. revenue plus BFD Ltd. Revenue less eliminations") and general relationships ("The write down allowance for an asset in any year after 1993 is limited to 25% of its original purchase price"). There has now been sufficient experimentation and implementation experience with XBRL for the exchange of business reporting information in live and planned applications to have illustrated both the need for an extension to the language to meet this need, as well as illustrating deficiencies in schemes used to date and the typical patterns of usage and the limitations on what that language actually needs to cover. This information is advantageous to have because it limits the scope and guides the design of the language.

1.1 Terminology and formatting conventions

Terminology used in XBRL frequently overlaps with terminology from other fields. The terminology used in this document is summarised in Table 1.

Table 1. Terminology

abstract element, bind, concept, concrete element, context, Discoverable Taxonomy Set (DTS), duplicate items, duplicate tuples, element, entity, equal, essence concept, fact, instance, item, least common ancestor, linkbase, period, taxonomy, tuple, unit, taxonomy schema, child, parent, sibling, grandparent, uncle, ancestor, XBRL instance, c-equal, p-equal, s-equal, u-equal, v-equal, x-equal, minimally conforming XBRL processor, fully conforming XBRL processor.	As defined in [XBRL].
MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, MAY, OPTIONAL	See [RFC2119] for definitions of these and other terms. These include, in particular: SHOULD — Conforming documents and applications are encouraged to behave as described. MUST — Conforming documents and consuming applications are required to behave as described; otherwise they are in error.
expression	An expression using constants, variables, arithmetic, logical, and functional operators.
formula	An expression along with criteria that indicate the domain of each variable in that expression and how they are to be bound.
rule	The term "rule" is not used in this version of the requirement, although it appeared previously as a synonym for "formula".
variable	A "variable" appears in expressions as a symbol that

	the application of a formula binds to a value, so that the expressions of that formula can be evaluated.
argument	An "argument" of a formula is a formal parameter bound to some portion of an input instance. Some of the arguments will be identified as variables to be used in expressions.

The following highlighting is used for positive examples:

Example 1. Example of an example

Counterexamples indicate incorrect or unsupported examples:

Example 2. Example of a counterexample

Selections from other normative documents is highlighted thus:

Example 3. Example of normative material

Non-normative editorial comments are denoted as follows and removed from final recommendations:

WH**:** This highlighting indicates editorial comments about the current draft, prefixed by the editor's initials.

Italics are used for rhetorical emphasis only and do *not* convey any special normative meaning.

Distinctively bracketed and coloured numbers {1.2.3} refer to that particular numbered section of the XBRL 2.1 Specification Recommendation [XBRL].

1.2 Normative status

This document is normative in the sense that any formula specification recommendations by XBRL International MUST satisfy the requirements as they are stated here.

This document version depends upon XBRL 2.1 Specification Recommendation [XBRL].

XBRL Specification 2.1 does *not* depend in any way upon this document.

1.3 Language independence

The official language of XBRL International's own work products is English and the preferred spelling convention is UK English. Unless otherwise stated, XBRL specifications must not require XBRL users to use English in documentation, item, tuple, entity, scenario or any other elements.

2 Use cases

The general use case for an XBRL formula specification is the externalisation and publication of a set of formulas that will be applied by a consuming application:

- A financial regulator collecting quarterly balance sheets and income statements;
- A statistical agency collecting market valuations of various asset categories;
- A tax authority accepting electronic tax filings;

- A stock exchange accepting registration requests;
- A bank evaluating loan applications or checking loan covenants;

More specifically, there are three general needs these consuming applicaitons may have:

1. Validating an XBRL instance and indicating success or failure. In this case, no XML or XBRL output is really needed.
2. Validating an XBRL instance and providing detailed error messages. In this case, general XML output—for further processing and rendering as appropriate by an application—would seem to be sufficient.
3. Transforming, adjusting, composing or creating XBRL instances based on existing XBRL instances. In this case, it is sensible for a formula processor to produce a valid XBRL instance. If producing applications verify that their own output will be accepted by those consuming applications, significant efficiencies are possible, particularly in a distributed environment such as the Internet.

These needs are not mutually exclusive, but are increasing level of ambition. Underlying all of the use cases below is the presumption that because level 3 above does subsume the other two, it is the level of functionality being sought. Therefore, the specific use cases driving the formula requirements are expressed as "before" and "after" fact sets and instances, along with a structured description of the relevant generalisation of the behaviour illustrated by that case.

The use cases below all depend on a single taxonomy shown in Figure 1 consisting mainly of financial position and performance items.

Figure 1. Items Appearing in Use Cases

@name	@type	@periodType	Label (en, standard)
Assets	monetaryItemType	instant	Assets
CurrentAssets	monetaryItemType	instant	Current Assets
FixedAssets	monetaryItemType	instant	Fixed Assets
Equity	monetaryItemType	instant	Equity
Shares	sharesItemType	instant	Shares
Price	monetaryItemType	instant	Price
Earnings	monetaryItemType	duration	Earnings
AvgShares	sharesItemType	duration	Average Shares
PE	pureItemType	instant	Price-Earnings Ratio
ROE	pureItemType	instant	Return on Equity
EPS	decimalItemType	duration	Earnings per Share
AssetsEquity	pureItemType	instant	Assets-to-Equity Ratio
Rating	integerItemType	instant	Rating
AssetsOkay	booleanItemType	instant	Assets Okay
EquityOkay	booleanItemType	instant	Equity Okay
AssetsMessage	stringItemType	instant	Assets Message
AssetsLB	booleanItemType	instant	Assets in Lower Bound
AssetsUB	booleanItemType	instant	Assets in Upper Bound
AssetsLarge	booleanItemType	instant	Assets not too Large
AssetsSmall	booleanItemType	instant	Assets not too Small
IntangiblesPatents	monetaryItemType	instant	Intangibles (Patents)
IntangiblesClass	stringItemType	instant	Intangibles Class
IntangibleGross	monetaryItemType	instant	Intangible Gross
IntangibleReserve	monetaryItemType	instant	Intangible Reserve
IntangibleNet	monetaryItemType	instant	Intangible Net
IntangibleAsset	tupleType		Intangible Asset
AutoWriteDown	tupleType		Automobile Write-Down
AutoName	tokenItemType	duration	Automobile Name
AcquisitionDate	dateItemType	duration	Acquisition Date
WriteDownAllowance	monetaryItemType	duration	Write-Down Allowance
WDVBroughtForward	monetaryItemType	duration	Write-Down Value Brought Forward
UsefulLife	durationItemType	duration	Useful Life

All facts in the use cases have a `unitRef` and `contextRef` drawn from the units and contexts shown in Figure 2 and Figure 4, respectively. The namespace prefix `si` refers to international standard scientific units.

Figure 2. Units Appearing in Use Cases

@id	measure
usd	iso4217:USD
gbp	iso4217:GBP
pure	xbrli:pure
shs	xbrli:shares
usdsh	iso4217:USD/xbrli:shares
gbpsh	iso4217:GBP/xbrli:shares
year	si:year

The namespace prefix `co` with elements `aaa` and `bbb` are used to distinguish segments of entity `444`. The elements `actual`, `budgeted` and `variance` are used to distinguish scenarios.

Figure 3. Segments and Scenarios Appearing in Use Cases

prefix	element
co:	<aaa/>
co:	<bbb/>
co:	<actual/>
co:	<budgeted/>
co:	<variance/>

Figure 4. Contexts Appearing in Use Cases

@id	entity / @identifier	entity / segment	period / instant	period / startDate	period / endDate	scenario
c01	333			2003-01-01	2003-12-31	
c02	333		2002-12-31			
c03	333		2003-12-31			
c04	444			2003-01-01	2003-12-31	
c05	444		2003-12-31			
c06	333		2003-12-31			
c07	444	<geo>ON</geo>		2003-01-01	2003-12-31	
c08	444	<geo>ON</geo>	2003-12-31			
c09	444	<geo>MI</geo>		2003-01-01	2003-12-31	
c10	444	<geo>MI</geo>	2003-12-31			
c11	444		2003-12-31			<co:actual/>
c12	444		2003-12-31			<co:budgeted/>
c13	444		2003-12-31			<co:variance/>
c14	444			1993-01-01	1993-12-31	
c15	444			1995-01-01	1995-12-31	
c16	333		2001-12-31			
c17	444	<lob>paper</lob>		2003-01-01	2003-12-31	
c18	444	<lob>paper</lob>	2003-12-31			
c19	444	<lob>plastic</lob>		2003-01-01	2003-12-31	
c20	444	<lob>plastic</lob>	2003-12-31			

Note that contexts `c06` and `c03` are s-equal.

The data type of the content in any `UsefulLife` fact is assumed to be `yearMonthDuration`; the XBRL `durationItemType` does not specify whether fact content is `yearMonthDuration` or `dayTimeDuration`.

Each fact in this document also has a unique identifier such as `f42`; the identifiers are not relevant in any way to processing the facts or formulas, but the identifiers allow them to be referred to in the text.

Expressions are written in XML Path Language 2.0 [XPATH2] and are meant to be transparent for the example at hand, not in and of itself prescriptive as to the expression language to be

required. Some cases use the syntax of XML Query Language 1.0 [XQuery] to declare functions; as stated in [XPATH2], "XPath is designed to be embedded in a host language such as XSLT 2.0 or XQuery… XPath per se does not provide a way to declare functions, but a host language may provide such a mechanism." The host language chosen in this example is XQuery because it allows users to declare functions of their own.

2.1 Formula uses items that are p-equal and u-equal in identical contexts

An important type of formula involves a mathematical operator applied to a pair of items when they are p-equal, c-equal and u-equal.

Formulas:

item	test	expression	type[2]	matching
Assets		$CurrentAssets + $FixedAssets	INF	p-equal, c-equal and u-equal

Facts:

@id	item	@contextRef	@unitRef	Precision	[content]
f33	CurrentAssets	c03	usd	INF	8000
f34	FixedAssets	c03	usd	INF	35000

Result:

@id	item	@contextRef	@unitRef	Precision	[content]
f13	Assets	c03	usd	INF	43000

2.2 Formula uses items that are p-equal, u-equal and c-equal

A variation of use case 2.1 above is when items are c-equal without the contexts being identical.

Formulas:

item	test	expression	type	matching
Assets		$CurrentAssets + $FixedAssets	INF	p-equal, c-equal and u-equal

Facts:

@id	item	@contextRef	@unitRef	Precision	[content]
f33	CurrentAssets	c03	usd	INF	8000
f35	FixedAssets	c06	usd	INF	35000

Result:

@id	item	@contextRef	@unitRef	Precision	[content]
f13	Assets	c03	usd	INF	43000

The output context is s-equal to c03 and c06. Formula processors should behave consistently and the specification must define which context to use on output.

2.3 Formula uses items that are p-equal, u-equal and contexts whose periods differ

Formulas must be able to express relations that cross periods. In this example the value of any two values for "Shares" that differ by one year are averaged during the intervening duration.

[2] "type" here means either the result type if non numeric, or else the value of the precision or decimals attribute when the result is numeric.

Formulas:

item	test	expression	type	matching
AvgShares		($SharesNext + $Shares) div 2	INF	p-equal, u-equal; "SharesNext" is in a context offset by one year from "Shares".

Facts:

@id	item	@contextRef	@unitRef	Precision	[content]
f05	Shares	c02	shs	INF	50000
f17	Shares	c03	shs	INF	60000

Result:

@id	item	@contextRef	@unitRef	Precision	[content]
f11	AvgShares	s-equal to c01	shs	INF	55000

The output context, because it is not present among the input facts, must be synthesised by the formula. In this example it happens to be s-equal to c01. Note that the two inputs are in "instant" contexts while the output is a "duration" context whose start and end dates are equal to those of the input instants (taking into account differences in the way startDate and endDate default the time of day {4.7.2}).

2.4 Formula uses items in contexts that match period endpoints

The matching of input periods may involve matching of endpoints. This example is a simple movement analysis on the value of Equity.

Formulas:

item	test	expression	type	matching
Equity		$EquityPrev + $Earnings	INF	p-equal, u-equal; "EquityPrev" refers to a context in the year previous to that of "Equity". The duration-type period of "Earnings" must begin at the instant represented by "EquityPrev".

Facts:

@id	item	@contextRef	@unitRef	Precision	[content]
f03	Equity	c02	usd	INF	17000
f09	Earnings	c01	usd	INF	9000

Result:

@id	item	@contextRef	@unitRef	Precision	[content]
f11	Equity	s-equal to c03	usd	INF	26000

More generally, endpoints may match in any relationship expressible by using comparison operators (equal, less than, greater than), arithmetic offsets (plus, minus), and constants (dates, datetimes, durations).

WH: Use cases with richer relative period expressions would be useful.

2.5 Expression yields a new unit of measure

The units of measure of inputs, when multiplied or divided, may yield a different unit of measure. The unit of measure need not have been previously defined in the input instance.

Formulas:

item	test	expression	matching
EPS		($Earnings div $AvgShares)	p-equal, c-equal.

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f36	Earnings	c01	usd	INF	11000
f11	AvgShares	c01	shs	INF	55000

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f29	EPS	c01	usdsh	INF	0.2

In this example there are no facts with the unit representing "USD per share"; it is synthesised by the formula for the output. In this example the precision of the output is infinite, just like the inputs.

2.6 Formula determines the result units of an expression from the units of the bound facts

The units of measure of outputs differ depending on the units of the input facts that were bound. If the result item is a numeric type then the formula specifies the result units as a function of the input units.

Formulas:

item	test	expression	type	matching
EPS		($Earnings div $AvgShares)	precision=9 and units(result) = units(Earnings) / units(AvgShares)	p-equal, c-equal.

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f09	Earnings	c01	usd	INF	9000
f10	Earnings	c01	gbp	INF	5000
f11	AvgShares	c01	shs	INF	55000

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f37	EPS	c01	usdsh	9	0.16363636
f38	EPS	c01	gbpsh	9	0.09090909

In this example the same formula computes results in the same context having different units depending on which of two input facts (f09 and f10) were used in the computation. The item specified in the result (e.g., EPS) determines the item type (e.g., `pureItemType`) and therefore constrains the possible units (e.g. pure) {4.8.2}.

Rejected requirement 10.6 below, "* Formulas MAY specify one or more alternative items or tuples as the possible result of expression evaluation," would allow a formula to produce either a decimal or a fraction as the result of a division, but without it the result will always be one or the other.

2.7 Formula overrides the natural result units of an expression

In this example the units of the Write-Down Allowance that would be determined from the inputs would be a currency amount "per year", but the formula must type cast the output to a

currency so as to agree with WriteDownAllowance which is a monetaryItemType and therefore must have a unit whose measure is an ISO currency type {4.8.2}.

Formulas:

item	test	expression	type	matching
WriteDownAllowance		$AcquisitionCost div fn:get-years-from-yearMonthDuration ($UsefulLife)	precision=INF and units(result) = units(AcquisitionCost)	p-equal, c-equal;

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f76	UsefulLife	c15			5Y
f78	AcquisitionCost	c15	gbp	INF	12000
f79	AcquisitionCost	c15	usd	INF	20000

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f80	WriteDownAllowance	c15	gbp	INF	2400
f81	WriteDownAllowance	c15	usd	INF	4000

The example shows that the currency (or any other part of the unit) cannot be “hard wired” into the formula but MUST be determined from the input facts.

2.8 Expression result has limited precision

When an expression containing the division operator uses a fact with precision="INF" as a divisor, and the result fact of the expression has a denominator with prime factors other than 2 and 5 the result is a repeating decimal without a finite representation.

The precision of a repeating decimal when not otherwise specified will be INF even though its lexical representation is limited to the number of digits of precision available on the executing hardware. This reflects the underlying reality of limited machine precision. Formula authors may choose to limit the precision still further. Moreover, if different processors were allowed to select their own output precision, formulas would yield different results on different machines. Therefore, formulas must be able to specify the desired precision of the output.

Formulas:

item	test	expression	type	matching
EPS		($Earnings div $AvgShares)	precision=8	p-equal, u-equal, c-equal.

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f09	Earnings	c01	usd	INF	9000
f11	AvgShares	c01	shs	INF	55000

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f37	EPS	c01	usdsh	8	0.1636363636363636363

The computation 9000/55000 yields a repeating decimal (.1636363…) whose lexical representation is limited to 18 digits by the default data type (double) of the executing processor.

2.9 Expression result has limited number of decimal places

Similarly to use case 2.6 above, the result may also be specified to a number of decimal places rather than by precision.

Formulas:

item	test	expression	type	matching
EPS		($Earnings div $AvgShares)	decimals=7	p-equal, u-equal, c-equal.

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f09	Earnings	c01	usd	INF	9000
f11	AvgShares	c01	shs	INF	55000

Result:

@id	item	@contextRef	@unitRef	@decimals	[content]
f38	EPS	c01	usdsh	7	0.1636364

2.10 Formulas may be prohibited

An extension {3.2} {3.5.3.9.7.5} of a set of formulas may need to change the method by which an item is computed in a base set of formulas. In this example, the simpler calculation of EPS is in the base set of formulas:

Formulas:

item	test	expression	type	matching
EPS		($Earnings div $Shares)	decimals=7	p-equal; Shares context period/instant = Earnings context period/endDate.

An extension set of formulas then prohibits that formula, and asserts a different one:

Formulas:

item	test	expression	type	matching
~~EPS~~		~~($Earnings div $Shares)~~	~~decimals=7~~	~~p-equal;~~ ~~Shares context period/instant =~~ ~~Earnings context period/endDate.~~
EPS		($Earnings div $AvgShares)	decimals=7	p-equal, c-equal.

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f36	Earnings	c01	usd	INF	11000
f11	AvgShares	c01	shs	INF	55000
f17	Shares	c03	shs	INF	60000

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f29	EPS	c01	usdsh	INF	0.2

In this example, the calculation based on average shares is used and only one value of EPS is calculated.

2.11 Incompatible formulas are distinguishable

Formulas maintained in the same set may provide alternative, possibly incompatible definitions for the same item. In this example, EPS for some purposes is computed using the

average number of shares during the earnings period, and for other purposes using the number of shares at the end of the period.

Formulas:

item	test	expression	type	matching
EPS	"method 1"	($Earnings div $AvgShares)	decimals=7	p-equal, c-equal.
EPS	"method 2"	($Earnings div $Shares)	decimals=7	p-equal; Shares context period/instant = Earnings context period/endDate.

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f36	Earnings	c01	usd	INF	11000
f11	AvgShares	c01	shs	INF	55000
f17	Shares	c03	shs	INF	60000

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f29	EPS	c01	usdsh	INF	0.2
f67	EPS	c01	usdsh	9	0.183333333

In this example, *only* one of the two results should be computed, depending on whether the (set of) formulas labelled "method 1" or "method 2" are selected for processing. The `xlink:role` attribute may be an appropriate way to model this, since the evaluation of the condition does not depend in any way on the facts in the instance.

2.12 Formula has a precondition on item values

Setting a precondition on an item value may be needed to ensure the expression is meaningful and will not cause an evaluation error.

Formulas:

item	test	expression	type	matching
PE	($EPS gt 0)	($Price div $EPS)	INF	p-equal, u-equal; Context of "Price" has a period that is the instant which ends that of "Earnings".

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f19	Price	c03	usdsh	INF	11.2
f29	EPS	c01	usdsh	INF	.2
f20	Price	c03	gbpsh	INF	5.2
f30	EPS	c01	gbpsh	INF	-0.8

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f21	PE	c03	pure	INF	56

This is an example in which the condition of one pair of facts (f19 and f29) evaluates to true, the other pair (f20 and f30) to false.

2.13 Formula applies only to certain time periods

Facts may match the formula only if they fall within a named time period. Note that because every item has a fixed `periodType`, it is *not* necessary to be able to specify the kind of period of which the fact is asserted, because that is implicit in the item itself.

Formulas:

item	test	expression	type	matching
AssetEquity	($Equity gt 0)	$Assets div $Equity	precision=10	p-equal, u-equal, c-equal; Context must have a period whose endpoint is strictly before 2003-01-01.

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f01	Assets	c02	usd	INF	100000
f03	Equity	c02	usd	INF	17000
f13	Assets	c03	usd	INF	43000
f15	Equity	c03	usd	INF	26000

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f31	AssetEquity	c02	pure	10	5.882352941

In this example, one pair of facts (f01 and f03) evaluates the condition to `true` and the other (f13 and f15) evaluates it to `false`.

2.14 Formula applies only to certain units

Facts may match the formula only if the units match.

Formulas:

item	test	expression	type	matching
AssetEquity	($Equity gt 0)	$Assets div $Equity	precision=10	p-equal, u-equal, c-equal; Assets context must have unit/measure = ISO4217:USD

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f01	Assets	c02	usd	INF	100000
f03	Equity	c02	usd	INF	17000
f02	Assets	c02	gbp	INF	43000
f04	Equity	c02	gbp	INF	26000

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f31	AssetEquity	c02	pure	10	5.882352941

In this example, one pair of facts (f01 and f03) evaluates the condition to `true` and the other (f02 and f04) evaluates it to `false`.

2.15 Formula applies only to the latest period in an instance

Facts may match the formula only if they fall within a time period that is described as the "latest" period in the instance. This is an extension of use case 2.13 above which allowed matching against a fixed time period.

Formulas:

item	test	expression	type	matching
AssetEquity	($Equity gt 0)	$Assets div $Equity	precision=18	p-equal, u-equal, c-equal; Context must have a period that is the latest occurring in the input instance.

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f01	Assets	c02	usd	INF	100000
f03	Equity	c02	usd	INF	17000
f13	Assets	c03	usd	INF	43000
f15	Equity	c03	usd	INF	26000

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f31	AssetEquity	c03	pure	18	1.653846153846153846

In this example, one pair of facts (f01 and f03) evaluates the condition to `false` and the other (f13 and f15) evaluates it to `true`. The latest period among the inputs is the instance 2003-12-31.

2.16 Expression contains a conditional

Conditional expressions and nested conditionals can test several conditions in sequence. In this example a continuous P/E ratio is mapped to one of three discrete values of Rating.

Formulas:

item	test	expression	type	matching
Rating		if ($PE gt 5) then 1 else if ($PE gt 10) then 2 else 3	INF	

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f21	PE	c03	pure	INF	56
f22	PE	c02	pure	INF	-6.5

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f39	Rating	c03	pure	INF	3
f40	Rating	c02	pure	INF	1

In principle, any conditional expression could be recast as a series of separate formulas with non-overlapping conditions (this example would require three formulas, with one having the condition "$5 \leq PE$ and $PE < 10$").

2.17 Precondition tests date items in historical period

Setting a precondition on an item value may be part of defining its scope of applicability.

In this example, historical data (Acquisition Cost, Useful Life, and Acquisition Date) of an asset determines its write down allowance for all future periods. Acquisitions made after 11 March 1992 are subject to a GBP 2000 maximum write-down per year; hence the acquisition date of 10 October 1993 means that after two years only GBP 4000 has been written down, not 2*(20000/6) had the acquisition been made (say) 1991.

In this example the context of the Write-Down Allowance is determined from the context of a nonzero Write-Down Value Brought Forward. The item `UsefulLife` is a `durationItemType` but the result of the expression is cast to the same type and unit as `AcquisitionCost`.

Formulas:

item	test	expression	type	matching
WriteDownAllowance	($AcquisitionDate lt 1992-03-11) and ($WDVBroughtForward gt 0)	$AcquisitionCost div fn:get-years-from-yearMonthDuration ($UsefulLife)	INF	p-equal, c-equal; Result context is that of a nonzero WDVBroughtForward whose period is greater than or equal to the input facts' end date.
WriteDownAllowance	($AcquisitionDate ge 1992-03-11) and ($WDVBroughtForward gt 0)	fn:min(2000, $AcquisitionCost div fn:get-years-from-yearMonthDuration ($UsefulLife))	INF	p-equal, c-equal; Result context is that of a nonzero WDVBroughtForward whose period is greater than or equal to the input facts' end date.

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f57	AutoWriteDown				(tuple)
f58	AssetName	c14			571XVH
f59	AcquisitionDate	c14			1993-10-10
f60	AcquisitionCost	c14	gbp	INF	20000
f61	UsefulLife	c14			6
f62	WDVBroughtForward	c15	gbp	INF	16000

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f63	WriteDownAllowance	c15	gbp	INF	2000

Although the formula requirements in this document do not presume the Financial Reporting Taxonomies Architecture [FRTA], nevertheless Section 2.6 of FRTA does suggest that all facts in a tuple instance will be c-equal. Hence this example encloses facts f58 through f62 within a single tuple (f57) even though they have different contexts, which may not occur often.

2.18 Formula matches facts in different segments

A fixed asset breakdown between an entity (444) and its segments (aaa and bbb) has the formula shown below, and the two figures 50,000 and 80,000 summing to 130,000.

The presumption of this use case is that if there is no segment element, this corresponds to the "universal" segment, i.e., the entire entity.

Formulas:

item	test	expression	type	matching
FixedAssets		$FixedAssetsaaa + $FixedAssetsbbb	INF	p-equal, u-equal; FixedAssets(aaa) and FixedAssets(bbb) contexts are s-equal except for the segment identifier (aaa and bbb, respectively).

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f41	FixedAssets	c08	usd	INF	50000
f42	FixedAssets	c10	usd	INF	80000

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f43	FixedAssets	c05	usd	INF	130000

Note that the XML elements aaa and bbb, which are to be used in the input instance, must be present in a namespace accessible to the formulas, just like the item names and other parts of a taxonomy.

2.19 Formula matches facts in different scenarios

The Earnings of an entity are reported as actual and budgeted in different scenarios, and in yet a third scenario the variance figure is to be computed from them.

There is no presumption in this use case is that if there is no scenario element that it means some kind of "universal" scenario; rather, scenario is simply unspecified.

Formulas:

item	test	expression	type	matching
Earnings(variance)		$EarningsActual – $EarningsBudgeted	INF	p-equal, u-equal; Earnings(actual), Earnings(variance) and Earnings(budgeted) contexts are s-equal except for the scenario identifier.

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f44	Earnings	c11	usd	INF	12000
f45	Earnings	c12	usd	INF	14000

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f46	Earnings	c13	usd	INF	-2000

Note that the XML elements actual, budgeted and variance, which are to be used in the input instance, must be present in a namespace accessible to the formulas, just like the item names and other parts of a taxonomy.

2.20 Formula produces a default fact in the absence of a matching fact

When Current Assets are known in a given context but Fixed Assets are not, a formula author may want to assert a default value (zero) for the item FixedAssets.

Formulas:

item	test	expression	type	matching
FixedAssets	$CurrentAssets and fn:not($FixedAssets)	0	INF	p-equal, u-equal and c-equal;

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f33	CurrentAssets	c03	usd	INF	8000

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f43	FixedAssets	c03	usd	INF	0

The conditions expressed in this formula would be evaluated sequentially in the sense that the formula only applies in a context where CurrentAssets is already bound. The function fn:not is

used as the predicate "unbound", in effect assuming that "unbound" variables are initially bound to the empty sequence until bound to something else by the formula processor".

Note also that fact f43 will appear in the output, not the input, so that any formulas that required input facts for CurrentAssets and FixedAssets in c-equal contexts would require additional formula processor iterations, as in use case 2.10 above.

2.21 Formula assumes default values in the absence of matching facts

When values are not known in a given context, a formula author may assume a default value (zero) for items, in this example, default values of zero for CurrentAssets and FixedAssets.

Formulas:

item	test	expression	type	matching
Assets		$CurrentAssets + $FixedAssets	INF	p-equal, c-equal and u-equal; If CurrentAssets unbound then CurrentAssets = 0; If FixedAssets unbound then FixedAssets = 0; If both are unbound the formula produces no result.

Facts:

@id	item	@contextRef	@unitRef	Precision	[content]
f33	CurrentAssets	c03	usd	INF	8000
f35	FixedAssets	c03	usd	INF	35000
f77	CurrentAssets	c02	gbp	INF	2000
f65	FixedAssets	c16	gbp	INF	5000

Result:

@id	item	@contextRef	@unitRef	Precision	[content]
f13	Assets	c03	usd	INF	43000
f64	Assets	c02	gbp	INF	2000
f66	Assets	c16	gbp	INF	5000

This set of facts covers each of the three possible binding cases in the formula.

2.22 Formulas ignore Nil facts by default but MAY bind them

Formula authors must have the choice whether or not to bind expressions with nil facts. At least for the purpose of describing formulas in the use cases we assume that the default is that nil facts may not be bound, and so nothing appears in the "test" column.

Formulas:

item	test	expression	type	matching
Assets		$CurrentAssets + $FixedAssets	INF	p-equal, c-equal and u-equal.
Rating	element(my:PE, xbrli:decimalItemType nillable)	if ($PE[@xsi:nil = "true"]) then 0 else if ($PE lt 5) then 1 else if ($PE lt 10) then 2 else 3	INF	

Facts:

@id	item	@contextRef	@unitRef	Precision	[content]
f68	CurrentAssets	c03	usd	INF	(xsi:nil)
f35	FixedAssets	c03	usd	INF	35000
f70	PE	c02	pure	INF	(xsi:nil)

Result:

@id	item	@contextRef	@unitRef	Precision	[content]
f71	Rating	c02	pure	INF	0

The Assets rule should produce no results. Furthermore, the formula had instead assigned a default value of zero to CurrentAssets then it should behave the same as use case 2.21 above. This suggests that use case 2.21 above represents a somewhat more robust treatment than use case 2.20 above, which would involve creating a new fact that would then be a duplicate {4.10} of the nil item.

The Rating rule should produce one fact as output.

2.23 Expression evaluation exceptions MAY be caught to produce a result

The expression language must support a try/catch or other exception handling mechanism and the formula allowed to produce a result. The following is only an example result and is not meant to be normative for all formulas.

Formulas:

item	test	expression	matching
EPS		Try ($Earnings div $AvgShares) Catch Exception Return <EPS xsi:nil="true"/>	p-equal, c-equal.

Facts:

@id	item	@contextRef	@unitRef	@precision	[content]
f36	Earnings	c01	usd	INF	11000
f72	AvgShares	c01	shs	INF	0

Result:

@id	item	@contextRef	@unitRef	@precision	[content]
f73	EPS	c01	usdsh	INF	(xsi:nil)

Note that the units, context and precision of the result remain the same as if the calculation had not thrown an exception.

2.24 Formula does not bind duplicate facts

In XBRL 2.1 {5.2.5.2} calculations are not performed when there are duplicate facts ("A calculation binds for a summation item if it has no duplicates in the XBRL instance…") or if the facts have nil values ("Items with nil values do not participate in calculation bindings"). The same approach applies consistently in formulas.

Formulas:

item	test	expression	type	matching
Assets		$CurrentAssets + $FixedAssets	INF	p-equal, c-equal and u-equal.

Facts:

@id	item	@contextRef	@unitRef	Precision	[content]
f33	CurrentAssets	c03	usd	INF	8000
f69	CurrentAssets	c03	usd	INF	10000
f35	FixedAssets	c03	usd	INF	35000

Result:

@id	item	@contextRef	@unitRef	Precision	[content]

The formula produces no results because f33 and f69 satisfy the XBRL 2.1 "duplicates" predicate. Furthermore, this case is distinct from the *absence* of `CurrentAssets` and therefore a formula specifying a default value (use case 2.21 above) should not apply.

2.25 Formula uses a locally defined symbolic constant

In this formula, `lowerTolerance` and `upperTolerance` are constants bound to "-500" and "500" respectively.

Formulas:

item	test	expression	type	matching
AssetsOkay		($lowerTolerance lt ($Assets – ($CurrentAssets + $FixedAssets)) and (($Assets – ($CurrentAssets + $FixedAssets) lt $upperTolerance)		p-equal, u-equal and c-equal;

Facts:

@id	item	@contextRef	@unitRef	Precision	[content]
f33	CurrentAssets	c03	usd	INF	8000
f35	FixedAssets	c03	usd	INF	35000
f13	Assets	c03	usd	INF	44000

Result:

@id	item	@contextRef			[content]
f48	AssetsOkay	c03			false

2.26 Formula uses a globally defined symbolic constant.

Here, `tolerance` is bound to "500" globally and so has the identical value in two different formulas, one for testing the upper bound (UB) and one for the lower bound (LB).

Formulas:

item	test	expression	type	matching
AssetsUB		($Assets – ($CurrentAssets + $FixedAssets)) lt $tolerance	Boolean	p-equal, u-equal and c-equal;
AssetsLB		$tolerance lt ($Assets – ($CurrentAssets + $FixedAssets))	Boolean	p-equal, u-equal and c-equal;

Facts:

@id	item	@contextRef	@unitRef	Precision	[content]
f33	CurrentAssets	c03	usd	INF	8000
f35	FixedAssets	c03	usd	INF	35000
f13	Assets	c03	usd	INF	44000

Result:

@id	item	@contextRef			[content]
f49	AssetsUB	c03			false
f50	AssetsLB	c03			true