

General-purpose Legal Mark-up Language (GLML)  
User Manual

(WORKING DRAFT)

2021

# 

**Contents**

[Introduction](#_2et92p0) **2**

[Background](#_sjzziq9jctu5) 2

[Purpose of this Manual](#_9vtupfui6wn4) 2

[Structuring Legal Documentation using GLML](#_4d34og8) **3**

[Technical Specifications](#_aqye4wf910bm) **4**

[Tags and Tag Data-Types](#_v3tm8jy8kak) 4

[Text](#_17dp8vu) 6

[Number](#_3rdcrjn) 7

[Percent](#_26in1rg) 8

[Yes/No](#_lnxbz9) 8

[Select One](#_35nkun2) 8

[Date](#_1ksv4uv) 9

[Currency](#_44sinio) 10

[Money](#_2jxsxqh) 10

[Multi Value Tags](#_3j2qqm3) 11

[Formatting and Display](#_ngtnciju72z5) 13

[Bold and Underline](#_bwss9du0r7qq) 13

[And & Or](#_qwae8remlf71) 13

[List, Bullet List & Number List](#_t4ekrthwoj0d) 14

[Utilising Tag Properties](#_3whwml4) 16

[Currency - Symbol & Description](#_2bn6wsx) 16

[Date - Month and Year](#_qsh70q) 16

[Multi Value - Count](#_3as4poj) 17

[Multi Value - Index](#_qzeanqaswulr) 17

[If Functions](#_lren46fwtax5) 19

[Introduction to If functions](#_adyekmxikoa6) 19

[If And](#_rt4guznkkc1o) 21

[If Or](#_vg23cz7eouds) 22

[Use of Count](#_dp04oj4wsv7b) 23

[References](#_32hioqz) **25**

# 

# Introduction

## 

## Background

General-purpose Legal Mark-up Language (**GLML**) is a mark-up language. A mark-up language is a type of programming language that uses ‘tags’ to structure elements of a document and make it machine readable. Because mark-up languages are also designed to be human-readable, they contain standard words and are more intuitive to the average user than typical programming syntax (such as that used in Python).[1]

The two most common mark-up languages are HTML and XML. HTML is used for structuring web pages so content can be displayed in a standard way across any browser. XML meanwhile is used for storing large quantities of data in a structured manner, so that it can be parsed (i.e. converted into a new format) and easily read by different software packages. Innovations in both HTML and XML enable multiple users (often across different locations and using different operating systems) to share, view and interact with content and data in a streamlined way.

In the legal space, various efforts have attempted to ‘structure’ contractual data (e.g. names of the parties, contract date etc.) in a standardised fashion, so that whilst the content and data in a document may change, it would remain easy to share, view and interact with. One such language is Legal XML. However, this and other attempts have more closely resembled a typical coding language, inhibiting adoption by the legal community.[2] Document generation using such languages is complicated to implement and generally requires a technological background. This requires lawyers to work with programmers and coders to structure and maintain documents, introducing operational risk.

Aimed at reducing complexity, GLML has been designed for use by lawyers and other market practitioners who do not have coding experience. GLML is built to allow lawyers to easily turn existing precedents and pro forma documentation into machine readable documents, which can be used to create digital workflows for legal contracts and provide for the seamless flow of structured data. Any document editor (including Microsoft Word) can be used to apply GLML to a document, allowing lawyers to create and maintain GLML template documents in the same way as they currently create and maintain their traditional documentation.

While the focus of this manual is on the application of GLML within contracts for capital markets transactions, GLML can be applied to any type of legal document.

## 

## Purpose of this Manual

This manual has been written for lawyers and other market practitioners. Its purpose is to provide an overview of GLML and introduce its technical specifications in an accessible manner. A lawyer without any prior expertise in technology or programming should be able to read this manual and feel comfortable with the fundamentals of GLML and how it can be applied to structure legal documents.

The first section ([*Structuring Legal Documentation using GLML*](#_4d34og8)) outlines how GLML fits in with the wider documentation process and the subsequent sections explain the different elements and functionality of GLML.

As GLML use increases over time, refinements and extensions will be made to its framework. It is therefore important to note that this manual is a “living” document that will be updated as GLML evolves.

# Structuring Legal Documentation using GLML

Perhaps without knowing it, most lawyers work with ‘structured’ legal documents almost every day. These come in the form of templates which have blank spaces, or ‘blobs’, for a draftsperson to complete with information and terms specific to the particular instance of that document. These templates might have logical statements written in footnotes, such as “Delete this section if X or Y”, and require square bracket searches to check outstanding information. Typically, a lawyer will manually input the relevant transaction terms into the template document in Microsoft Word or other word processor and the document may be reviewed and/or negotiated over a series of email or telephone exchanges in a process that can be slow and prone to errors. Once the document is complete, the terms contained in that document exist only as characters in Microsoft Word or pixels in a PDF document - they need to be read by a human to be extracted and understood.

Using GLML, legal template documents can be turned from an analogue into a digital structure.

This is done by a lawyer or market practitioner (who we refer to as the draftsperson in this manual) “marking-up” the traditional legal template document with GLML tags and syntax (as explained in the sections below), turning it into a “**GLML template**”. Once marked-up with GLML, a document becomes machine readable. The GLML template can then be uploaded into an appropriate “**compiler**” software that is capable of reading the GLML template and using it to create a digital “**workflow**” where an end-user can input the transaction terms. In the workflow, the end-user is essentially assigning values to the tags that appear in the GLML template. The compiler can then produce a “**compiled document**” where the values inputted in the workflow replace the tags in the GLML template. By virtue of being captured digitally, these values are capable of being reviewed and negotiated in a structured manner as well as being seamlessly transferred to any connected downstream processes.

The GLML examples used in this manual show the GLML template view, the workflow view and the compiled document view to highlight the different stages described above.

|  |
| --- |
| ***GLML Template*** *(This cell shows the text of a GLML Template document) E.g.:*  The security shall be issued by ${Issuer}. |
| ***Workflow***  *(This cell shows how the user assigns values to tags in the Workflow. Yellow highlighted spaces indicate where the user may type/select values. Defined values are also presented highlighted in yellow)* *E.g.:*  Issuer: [ ]  (User assigns value to tag by typing entry)  Issuer: [Examples!123£ plc] |
| ***Compiled Document*** *(This cell shows the view of the Compiled Document, that is how the end-product document will look based on the GLML Template and the values assigned within the Workflow) E.g.:*  The security shall be issued by Examples!123£ plc. |

# Technical Specifications

## Tags and Tag Data-Types

What is a tag?

As discussed above, template legal documents typically contain various fields for a draftsperson to ‘fill in’. These fields usually exist in square brackets. In a bond issuance process for example, the nominal value of the bond will likely change from one issuance to the next - i.e. it is a *variable*. In a template, the names of such variables might appear in the left hand side column of a table, with their relevant value - or a placeholder for their value - inserted to the right, or they might appear within a paragraph of text. Illustrative examples are shown below.

|  |  |
| --- | --- |
| **Table** | |
| Issuer | [Example A plc] |
| Joint Lead Managers | [JLM A], [JLM B], [JLM C] |
| Nominal Amount | £[*insert nominal value*] |
| **Paragraph** | |
| The Base Prospectus dated [*base prospectus date*] has been prepared in accordance with the Prospectus Regulation. | |

In GLML, variables are called ‘tags’. A tag can contain either a single value or list of multiple values. In the above example, the Nominal Amount and Joint Lead Managers fields would give rise to single and multi value tags respectively.

Once a GLML template has been read by a compiler, an end-user can assign values to the tags in that GLML template in the workflow. Everywhere within a document that a tag appears, it will be replaced by the same value (or list of values) when the document is compiled.

How are tags written in GLML?

*For visual ease, all GLML syntax is highlighted in yellow in this manual. There is no need for GLML syntax to be highlighted in GLML templates.*

All tags follow a standard naming convention. They are written as ${tag}. Tags containing multiple words are separated by an underscore: ${tag\_name}. No punctuation other than the underscore between words can be included.

The “$” and curly bracket components make the tag machine readable. Some example tags are shown below.

|  |  |
| --- | --- |
| **Plain text** | **Tag** |
| Issuer | ${Issuer} |
| Joint Lead Managers | ${Joint\_Lead\_Managers} |
| Nominal Amount | ${Nominal\_Amount} |

What comprises a tag?

A tag comprises four components: ‘key’, ‘label’, ‘data-type’ and ‘value’.

* A **key** can be simply described as the tag written in GLML form (e.g. the key for ‘Nominal Amount’ is just ‘Nominal\_Amount’).
* The **label** is how a tag appears to an end user in the workflow i.e. it is the prompt for the end user to input the relevant value. It is often the plain text version of the key. For simple tags (e.g. ${Issuer}), it is often unnecessary to expand on the tag in the label because it is clear what information is required of the end user. For other fields, it might be preferable to expand the label to explain what input is required. Two examples are shown below.

|  |  |  |
| --- | --- | --- |
| **Plain text** | **Key** | **Label** |
| Early Redemption Event | Early\_Redemption\_Event | Please specify the Early Redemption Event |
| Joint Lead Managers | Joint\_Lead\_Managers | Joint Lead Managers |

* Tags must also have a **type**. Within a legal document there might be multiple forms of data. In capital market issuances for example, Final Terms documents contain currencies, dates, monetary values, percentage figures and free-standing text. These forms of data are known as ‘data-types’ and each tag must be assigned one. At the end of a GLML template document, a “**Data Table**”lists all of the tags in that document; it is here where the draftsperson allocates each tag a data-type. The data-types are explained more fully in the sub-sections below, but some examples are shown here.

|  |  |  |
| --- | --- | --- |
| **Key** | **Label** | **Type** |
| Early\_Redemption\_Event | Please specify the Early Redemption Event. | text |
| Issuance\_Currency | Issuance Currency | currency |
| Maturity\_Date | Maturity Date | date |
| Underwriting\_Commission | What is the underwriting commission on this issuance? | percent |

When a compiler reads the GLML template, the compiler will look to the Data Table for instruction on what type of data must be inputted in the workflow for a relevant tag. This is important. If a tag is allocated an incorrect data-type, the end user will not be able to input the appropriate value i.e. the end user could encounter a field which only accepts numerical values when trying to insert the Issuer name.

* Finally, each tag must be assigned a **value** that is applied to a particular tag (e.g. ‘3.00’). This can be set in the GLML template document itself as a default by specifying the value in the Value column of the Data Table (in which case the default value will pre-populate in the workflow). Alternatively it can be left blank (in which case the end user will need to specify the value in the workflow).

Data Table:

The Data Table must appear at the end of the GLML template and specify the key, label, type and default value (if any) for every tag that appears in the body of the GLML template. An example is shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Label** | **Type** | **Value** |
| Early\_Redemption\_Event | Please specify the Early Redemption Event | text |  |
| Issuance\_Currency | Issuance Currency | currency | Euro |
| Maturity\_Date | Maturity Date | date |  |
| Underwriting Commission | What is the underwriting commission on this issuance? | percent |  |

The GLML examples in the sub-sections below also show how the tags used in each example are expressed in the data table.

Data-type

The purpose of restricting tags to meet certain predefined data-types are twofold:

* Preventing a completely erroneous value from being assigned to a tag type reduces the likelihood of an error being made in the document’s production, and will reduce checking time. For example, an end user cannot accidentally give a tag the value “Eight pounds” if it has a Money data-type. Instead, the end user will only be able to assign it values of the form “8.00” (in this instance a Currency tag would also be used to specify the currency).
* Instructions as to the format of certain tags allow for straightforward computation. This is most obvious in instances where a tag is used in a calculation: predefining a tag to have a Number data-type allows its value to be read as a number rather than as text, making calculations possible.

The different data-types are outlined below.

### **Text**

Tags with the Text data-type are the least restrictive. Any character (including punctuation and spaces) or series of characters can form the value of a Text data-type tag.

|  |
| --- |
| ***GLML Template***  The security shall be issued by ${Issuer}. |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Issuer | Issuer | Text |  | |
| ***Workflow***  Issuer: [ ]  *(User assigns value to tag by typing entry)*  Issuer: [Examples!123£ plc] |
| ***Compiled Document***  The security shall be issued by Examples!123£ plc. |

### **Number**

Tags with the Number data-type only accept numeric values as entries.

|  |
| --- |
| ***GLML Template***  There are ${Number\_of\_existing\_tranches} Existing Tranches of this security. |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Number\_of\_existing\_tranches | Number of Existing Tranches | number |  | |
| ***Workflow***  Number of Existing Tranches: [ ]  *(User assigns value to tag by typing entry)*  Number of Existing Tranches: [1r ]  *(Non-numeric character is deleted in real-time and user continues to enter correct value)*  Number of Existing Tranches: [13] |
| ***Compiled Document***  There are 13 Existing Tranches of this security. |

Number data-type tag values do not have to be whole numbers. The data-type also accepts decimals.

|  |
| --- |
| ***GLML Template***  Each share represents ${Single\_Share\_Vote} votes. |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Single\_Share\_Vote | Single Share Vote | number | 0.8 | |
| ***Workflow***  Single Share Vote: [0.8]  *(Value is defaulted in template and the default value pre-populates in the workflow, user may elect to change by re-typing)*  Single Share Vote: [0.9] |
| ***Compiled Document***  Each share represents 0.9 votes. |

### **Percent**

Tags with the Percent data-type only accept numeric characters in their entry. As with numbers, the Percent data-type tag values do not need to be whole numbers; decimals are accepted.

|  |
| --- |
| ***GLML Template***  The Final Redemption Amount will be ${Final\_Redemption\_Amount}% of the Nominal Amount. Interest will be paid at a fixed rate of ${Fixed\_Interest\_Rate} per cent. per annum. |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Final\_Redemption\_Amount | Final Redemption Amount | percent |  | | Fixed\_Interest\_Rate | Fixed Interest Rate | percent |  | |
| ***Workflow***  Final Redemption Amount (%): [ ]  Fixed Interest Rate (%): [ ]  *(User assigns value to tags by typing entry)*  Redemption Amount (%): [100]  Fixed Interest Rate (%): [2.125] |
| ***Compiled Document***  The Final Redemption Amount will be 100% of the Nominal Amount.  Interest will be paid at a fixed rate of 2.125 per cent. per annum. |

### **Yes/No**

Tags with the Yes/No data-type may only take one of two values: Yes or No. Such tags are most widely used in If functions to determine whether or not to insert certain paragraphs and sections of a legal document (see [*If Functions*](#_lren46fwtax5) for further information). Selection of the Yes or No options is identical to the selection of an option for Select One data-type tags, described below.

### **Select One**

Select One data-type tags are one of the most restrictive and are used only when a finite number of values may be given to the tag. For these tags, the end user can select from the predefined options in the workflow. The benefits of using predefined options are that it improves readability and accessibility. It also allows data points to be presented in a uniform format across legal documentation. Where selectables are used in a document, a “**Selectables Table**” is required to set out all relevant options in respect of each Select One data-type tag. The Selectable Table follows the Data Table at the bottom of a GLML template document.

|  |
| --- |
| ***GLML Template***  The Day Count Fraction will be ${Fixed\_Rate\_Day\_Count\_Fraction} |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Fixed\_Rate\_Day\_Count\_Fraction | Fixed Rate Day Count Fraction | select\_one |  |   ***Selectables Table***   |  |  | | --- | --- | | **Name** | **Label** | | Fixed\_Rate\_Day\_Count\_Fraction | 30/360 | | Fixed\_Rate\_Day\_Count\_Fraction | Actual/Actual (ICMA) | | Fixed\_Rate\_Day\_Count\_Fraction | Actual/360 | |
| ***Workflow***  Fixed Rate Day Count Fraction: [30/360] / [Actual/Actual (ICMA)] / [Actual/360]  *(User assigns value to tag by clicking on option)*  Fixed Rate Day Count Fraction: [30/360] |
| ***Compiled Document***  The Day Count Fraction will be 30/360 |

### **Date**

Date data-type tags allow the user to select a date in the “DD Month YYYY” format in the workflow. Specific properties and formats of date are sometimes used in documents (for example, when discussing the maturity date of a bond, it might only be necessary to specify the month and year of maturity). The process for formatting Date tags is detailed in [*Utilising Tag Properties*](#_3whwml4).

|  |
| --- |
| ***GLML Template***  The security will be issued on ${Issue\_Date} |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Issue\_Date | Issue Date | date |  | |
| ***Workflow***  Issue Date*:* [*Calendar for date entry*]  *(User assigns value to tags by selecting date)*  Issue Date: [20 March 2020] |
| ***Compiled Document***  The security will be issued on 20 March 2020 |

### **Currency**

Currency data-type tags are similar to the Select One data-type. The main difference between them is that different elements of a Currency tag may be utilised. This is because each Currency tag encapsulates a letter code, a currency symbol, and a full name description for the currency (e.g. USD, $, United States Dollars). See [*Currency - Symbol & Description*](#_2bn6wsx) for further information. The currencies available for selection need to be included in the Selectables Table.

|  |
| --- |
| ***GLML Template***  The Specified Currency shall be ${Issuance\_Currency} |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Issuance\_Currency | Issuance Currency | currency |  |   ***Selectables Table***   |  |  | | --- | --- | | **Name** | **Label** | | Currency | EUR | | Currency | GBP | | Currency | USD | |
| ***Workflow***  Issuance Currency: [GBP] / [USD] / [EUR]  *(User assigns value to tag by clicking on a currency)*  Issuance Currency: [EUR] |
| ***Compiled Document***  The Specified Currency shall be EUR |

### **Money**

Money data-type tags are linked to a pre-specified Currency tag. Money tags are linked to Currency tags in the *Type* column of the Data Table. The link is established through a backslash (e.g. “money/Issuance\_Currency”). Money data-type tags are similar to Number data-type tags in that only numerical digits and decimal values may be entered.

The example below demonstrates how Currency and Money data-type tags can be used together. Within the example, the Currency’s symbol is called. See [*Utilising Tag Properties*](#_3whwml4) for further details.

|  |
| --- |
| ***GLML Template***  Expenses for Admission to a Trading Venue are expected to be ${Trading\_Admission\_Expenses\_Currency; Symbol}${Trading\_Admission\_Expenses\_Estimate} |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Trading\_Admission\_Expenses\_Currency | Trading Admission Expenses Currency | currency |  | | Trading\_Admission\_Expenses\_Estimate | Trading Admission Expenses Estimate | money/Trading\_Admission\_Expenses\_Currency |  | |
| ***Workflow***  Trading Admission Expenses Currency: [GBP] / [USD] / [EUR]  *(User assigns value to tags by clicking on a currency)*  Trading Admission Expenses Estimate (GBP): [10e ]  *(Non-numeric character is deleted in real-time and user continues to enter correct value)*  Trading Admission Expenses Estimate (GBP): [1,000]  *(Comma inserted automatically)* |
| ***Compiled Document***  Expenses for Admission to a Trading Venue are expected to be £1,000 |

### **Multi Value Tags**

Some tags represent data points with multiple values. Instead of the tag representing a single value, a Multi Value tag is capable of holding a list of values. In the compiled document, the tag will be replaced with a list of all of the values specified. Multi Value tags still have a data-type with the same rules as above as to what can and cannot be entered in a value. Currently only Text data-type lists and Date data-type lists are supported.

Having such tags as lists of separated data entries allows for different kinds of formatting and usage, such as inserting “and” between the last two entries in a list. These formatting properties are discussed in [*Formatting and Display*](#_ngtnciju72z5) below.

An example use of a Multi Value tag is shown below. The order of the values within the tag is preserved throughout the tag’s use in the document, and is identical to the order in which the values were entered in the workflow.

|  |
| --- |
| ***GLML Template***  The managers in the syndicate are: ${Managers}. |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Managers | Managers | text\_list |  | |
| ***Workflow***  Managers: [ ]  *(User types first entry)*  Managers: [Bank 1 plc]  *(User presses ‘Enter’)*  Managers: [Bank 1 plc], [ ]  *(User types additional entries, pressing ‘Enter’ after each)*  Managers: [Bank 1 plc], [Bank 2 Ltd], [Bank 3 Ag] |
| ***Compiled Document***  The managers in the syndicate are: Bank 1 plc, Bank 2 Ltd, Bank 3 Ag. |

Date data-type lists work in the same way. The user selects dates from a calendar in the workflow and each selected date is added to the list of values.

|  |
| --- |
| ***GLML Template***  The Offering Circular and the supplements to it dated ${Supplement\_Dates} form a prospectus. |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Supplement\_Dates | Supplement Dates | date\_list |  | |
| ***Workflow***  Supplement Dates: [*Calendar for date entry*]  *(User assigns value to tag by selecting dates in calendar)*  Supplement Dates: [2 March 2018], [5 June 2018], [8 September 2018] (*based on calendar entries*) |
| ***Compiled Document***  The Offering Circular and the supplements to it dated 2 March 2019, 5 June 2019, 8 September 2019 form a prospectus. |

## 

## Formatting and Display

How data is displayed in a compiled document can be modified by altering the tag in the underlying GLML template. This may involve directly changing the tag format (such as for bold and underlining), or by adding a semi-colon and instructive text before the tag is closed. This is most relevant for Multi Value tags but simple formatting changes can also be made to single value tags.

### **Bold and Underline**

Bold or underline formatting can be achieved in the compiled document by making the tag itself bold or underlined (as applicable) in the template.

*Please note that if tags are highlighted in the GLML template then the highlighting will be retained in the compiled document when the relevant value replaces the tag.*

|  |
| --- |
| ***GLML Template***  The security will be guaranteed by **${Guarantor}**.  The syndicate consists of ${Managers} acting as managers. |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Guarantor | Guarantor | text |  | | Managers | Managers | text\_list |  | |
| ***Workflow***  Guarantor: [Guarantor Ltd]  Managers: [Bank 1 plc], [Bank 2 Ltd], [Bank 3 Ag] |
| ***Compiled Document***  The security will be guaranteed by **Guarantor Ltd**.  The syndicate consists of Bank 1 plc, Bank 2 Ltd, Bank 3 Ag acting as managers. |

### **And & Or**

The And and Or formats are specific to Multi Value type tags (e.g. text\_list and date\_list) and allow for And or Or to be inserted between the last two values of a list. If a list happens to only have one value then it is simply that single value that appears in the compiled document.

|  |
| --- |
| ***GLML Template***  The managers in the syndicate are ${Managers; And}.  Business days will be days on which one of ${Business\_Centres; Or} are open.  The security is not to be marketed in ${Restricted\_Regions; And}. |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Business\_Centres | Business Centres | text\_list |  | | Managers | Managers | text\_list |  | | Restricted\_Regions | Restricted Regions | text\_list |  | |
| ***Workflow***  Managers*:* [Bank 1 plc], [Bank 2 Ltd], [Bank 3 Ag]  Business Centres: [London], [Frankfurt]  Restricted Regions: [China] |
| ***Compiled Document***  The managers in the syndicate are Bank 1 plc, Bank 2 Ltd and Bank 3 Ag. Business days will be days on which one of London or Frankfurt are open.  The security is not to be marketed in China. |

### **List, Bullet List & Number List**

The List, Bullet List and Number List can be used in respect of Multi Value type tags only. These facilitate a vertical list of the values, with each value on a new line. Bullet List has the additional feature of placing a bullet point before each value, while Number List places an integer value in front of each entry starting with 1.

|  |
| --- |
| ***GLML Template***  The managers in the syndicate are:  ${Managers; List}  The business centres are:  ${Business\_Centres; Bullet List}  The restricted regions are: ${Restricted\_Regions; Number List} |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Business\_Centres | Business Centres | text\_list |  | | Managers | Managers | text\_list |  | | Restricted\_Regions | Restricted Regions | text\_list |  | |
| ***Workflow***  Managers: [Bank 1 plc], [Bank 2 Ltd], [Bank 3 Ag]  Business Centres: [London], [Frankfurt], [Singapore]  Restricted Regions: [China], [United States of America], [South Korea] |
| ***Compiled Document***  The managers in the syndicate are:  Bank 1 plc Bank 2 Ltd Bank 3 Ag  The business centres are:   * London * Frankfurt * Singapore   The restricted regions are:   1. China 2. United States of America 3. South Korea |

## Utilising Tag Properties

As well as being able to display certain tags in different formats, a draftsperson is able to utilise properties of certain tag data-types. For example, within a date data-type tag a draftsperson may only wish to specify the month and year of bond maturity (as opposed to the full date). Similar to tag formatting, a draftsperson utilises these properties by using keywords within the curly brackets of the tag.

### **Currency - Symbol & Description**

Currency data-type tags have three different ways of being utilised:

* *Default* - by default, a Currency tag is replaced by the three-letter short code for the currency. For example, “${Issuance\_Currency}” will display the currency in the form of XXX (e.g. EUR).
* *Symbol* - instead of displaying the currency in XXX format, a draftsperson may wish to only include the symbol (e.g. €) of the relevant currency. In such a case, the draftsperson adds “; Symbol” directly after the relevant tag wording. The GLML syntax for this example would now read “${Issuance\_Currency; Symbol}”.
* *Description* - a user may also wish to use the currency ‘Description’ (e.g. Euro). In this case “Description” replaces “Symbol” in the above example i.e. “${Issuance\_Currency; Description}”.

|  |
| --- |
| ***GLML Template***  The specified currency will be ${Issuance\_Currency; Description} (“${Issuance\_Currency}”)  The nominal amount is ${Issuance\_Currency; Symbol}${Nominal\_Amount} |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Issuance\_Currency | Issuance Currency | currency |  | | Nominal\_Amount | Nominal Amount | money/Issuance\_Currency |  | |
| ***Workflow***  Issuance Currency: [GBP]  Nominal Amount (GBP): [10,000,000] |
| ***Compiled Document***  The specified currency will be Pounds sterling (“GBP”)  The nominal amount is £10,000,000 |

### **Date - Month and Year**

Within the Date data-type, the month and/or the year can be utilised instead of the full date associated with a tag. The syntax utilising these properties is analogous to that seen for currencies above.

|  |
| --- |
| ***GLML Template***  The Notes will mature in ${Maturity\_Date; Month} ${Maturity\_Date; Year}. |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Maturity\_Date | Maturity Date | date | 5 July 2025 | |
| ***Workflow***  Maturity Date: [5 July 2025] |
| ***Compiled Document***  The Notes will mature in July 2025. |

### **Multi Value - Count**

Multi Value tags (e.g. text\_list and date\_list) have a ‘Count’ property. The Count feature instructs a compiler to replace a list of values with the number of values in that list.

|  |
| --- |
| ***GLML Template***  The managers in the syndicate are: ${Managers}  There are ${Managers; Count} managers.  There are ${Guarantors; Count} guarantors. |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Guarantors | Guarantors | text\_list |  | | Managers | Managers | text\_list |  | |
| ***Workflow***  Managers: [Bank 1 plc], [Bank 2 Ltd], [Bank 3 Ag]  Guarantors: [Guarantor plc], [Guarantor Ltd] |
| ***Compiled Document***  The managers in the syndicate are: Bank 1 plc, Bank 2 Ltd, Bank 3 Ag  There are three managers.  There are two guarantors. |

### **Multi Value - Index**

Multi Value tags (e.g. text\_list and date\_list) also have an ‘Index’ property. The Index property instructs a compiler to return a specific value (e.g. a name or date) from within a list of values. The value inserted reflects the position of that value in the list of values for that tag (i.e. the order in which the values are input during the workflow).

The syntax takes the form of tag\_name followed by a semi-colon and the relevant instruction. For example, if a draftsperson wanted to utilise the second value from a list, the GLML entry would take the form of “${Tag\_Name; 2}”. Further examples are shown below.

|  |
| --- |
| ***GLML Template***  The Managers are ${Managers; And}. The Lead Manager is ${Managers; 1}. The Co-Managers are ${Managers; 2} and ${Managers; 3}. |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Managers | Legal Name(s) of the Manager(s) | text\_list |  | |
| ***Workflow***  Legal Name(s) of the Manager(s): [Manager A], [Manager B], [Manager C] |
| ***Compiled Document***  The Managers are Manager A, Manager B and Manager C. The Lead Manager is Manager A. The Co-Managers are Manager B and Manager C. |

## 

## If Functions

### **Introduction to If functions**

The If function is a key component of GLML which checks whether a ‘condition’ is true or false. Such functions are primarily written into a template to make the insertion of particular sections of text (or GLML tags or further functions) dependent on whether a particular condition is true (e.g. to insert the floating rate section of a Final Terms if the relevant bond has floating rate interest). For example:

$${If ${Interest\_Basis} =[Floating Rate] ;Insert}{The Notes will have floating rate interest}

This allows a single GLML template to be used for a wide variety of value combinations.

*The examples in this manual include the If function syntax highlighted in green for visual ease. There is no need to include green highlighting in GLML templates.*

The syntax for the If function is as follows:

$${If X Condition Y ; Insert}{*Text to be inserted*}

An If function is initiated in the document with a double-dollar ($$) sign followed by an open-curly bracket and “If”. The condition then follows. X and Y may both be tags or one may be a tag and the other a set value/text (the floating rate example above and the examples below show X as a select-one type tag and Y being a selectable). The word ‘Insert’ is then included after a semicolon and before a close-curly bracket. This is followed by a second set of curly brackets which contains the text, GLML tag or further If functions that are to be inserted into the compiled document if the condition is found to be true.

When the compiled document is rendered following user input during the workflow, the contents of the second curly brackets i.e. ‘*Text to be inserted*’ will only appear in the compiled document where the condition is true. If the result is false, neither the text nor any If function will appear in the compiled document.

The examples in this section include a ‘System Function Interpretation’ row to explain how the If function logic is applied by a compiler.

Conditions can take on the forms outlined below:

**Equals / not equals**

* ‘=’ an **equals** will be true if X = Y. For example, if a user wanted to insert specific text if the interest basis (X) was floating rate (Y), the GLML might read:

$${If ${Interest\_Basis} = [Floating Rate] ; Insert}{The Interest Basis will be Floating Rate.}

* ‘NOT=’ a **not equals** will be true where X does not = Y. For example, to avoid having to input interest payment dates in the instance of the interest basis (X) being zero coupon (Y), the GLML might read:

$${If ${Interest\_Basis} NOT=[Zero Coupon] ; Insert}{${Interest\_Payment\_Dates}}.

In this example, a further GLML tag is inserted rather than plain text.

**Greater than / less than**

* ‘>’ a **greater than** tag will be true where X is greater than Y. Both X and Y must be Number or Money type tags. Alternatively, they can be Multi Value tags with the Count property utilised, or Date type tags with the Year property utilised. For example, $${If ${Lead\_Managers; Count} >3 ;Insert}{There are more than three lead managers}.
* ‘<’ a **less than** tag will be true if X is less than Y. X and Y need to be Number or Money type tags. As above, they can also be Multi Value tags with the Count property utilised, or Date type tags with the Year property utilised. For example, $${If ${Maturity\_Date; Year} <2045 ;Insert}{The bond matures before 2045}.

A full example, using the ‘=’ condition, is given below. This example also includes an If function within the curly brackets of another If function.

|  |
| --- |
| ***GLML Template***  $${If ${Interest\_Basis} = [Floating Rate] ; Insert}{The Interest Basis will be Floating Rate.}  $${If ${Ratings\_Required} = [yes] ; Insert}{  $${If ${Moodys\_Rating\_Required} = [yes] ; Insert}{The Notes have been rated ${Moodys\_Rating} by Moody’s.} } |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Interest\_Basis | Interest Basis | select\_one |  | | Ratings\_Required | Have the Instruments been rated? | yes\_no |  | | Moodys\_Rating\_Required | Is a Moody’s rating required? | yes\_no |  | | Moodys\_Rating | Moody’s Rating | text |  |   ***Selectables Table***   |  |  | | --- | --- | | **Name** | **Label** | | Interest\_Basis | Fixed Rate | | Interest\_Basis | Floating Rate | |
| ***Workflow***  Interest Basis: [Fixed Rate]  Have the Instruments been rated?: [Yes]  Has a rating been provided by Moody’s?: [Yes]  Moody’s Rating: [Aaa] |
| ***System Function Interpretation***  First Statement:   * Compiler checks the values of the tags within the functional statement * If Interest\_Basis = [Floating Rate] then the following text is inserted: “The Interest Basis will be Floating Rate.” * If Interest\_Basis = [Fixed Rate] then the text is not inserted. * Fixed Rate is selected by the user in the workflow so the text is not inserted in the compiled document.   Second Statement:   * Compiler checks the values of the tags within the functional statement * If Ratings\_Required is [Yes] then the compiler moves on to the next functional statement * If Ratings\_Required is [No] then nothing is inserted in the document and the subsequent functional statement is ignored. * Yes is selected by the user in the workflow so the compiler proceeds to the next functional statement   Third Statement:   * Compiler checks the values of the tags within the functional statement * If Moodys\_Rating\_Required = [Yes] then the following text and GLML tag is inserted “The Notes have been rated ${Moodys\_Rating} by Moody’s.” * If Moodys\_Rating\_Required = [No] then no text or further tag is inserted * Yes is selected by the User and Aaa is input as the Moody’s rating so this is inserted into the compiled document. |
| ***Compiled Document***  The Notes have been rated Aaa by Moody’s. |

### **If And**

The ‘If And’ function builds on the If function. Put simply, If And tests two sets of statements. If both tests return true, then the function executes the action (i.e. inserts whatever is between the next pair of curly brackets). If either or both tests return false, then the action will not be executed. The syntax for If And is similar to the If function in that it is opened in the same way with the word “Insert” appearing at the end. The only difference is the word “AND” between the two tests.

The conditions and allowed forms of the arguments remain the same as in the If function.

|  |
| --- |
| ***GLML Template***  $${If ${S\_And\_P\_Rating\_Required} = [yes] AND ${Moodys\_Rating\_Required} = [yes] ; Insert}{The Notes have been rated by both S&P and Moody’s}  $${If ${Interest\_Basis} = [Floating Rate] AND ${Interest\_Rate\_Determination} = [Screen Rate] ; Insert}{The Interest Rate will be a Floating Rate calculated using a Screen Rate} |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Interest\_Basis | Interest Basis | select\_one |  | | Interest\_Rate\_Determination | Interest Rate Determination | select\_one |  | | S\_And\_P\_Rating\_Required | Is an S&P rating required? | yes\_no |  | | Moodys\_Rating\_Required | Is a Moody’s rating required? | yes\_no |  |   ***Selectables Table***   |  |  | | --- | --- | | **Name** | **Label** | | Interest\_Basis | Fixed Rate | | Interest\_Basis | Floating Rate | | Interest\_Rate\_Determination | Screen Rate | | Interest\_Rate\_Determination | ISDA | |
| ***Workflow***  Is an S&P rating required?: [Yes]  Is a Moody’s rating required?: [Yes]  Interest Basis: [Floating Rate]  Interest Rate Determination: [ISDA Determination] |
| ***System Function Interpretation***  If BOTH S\_And\_P\_Rating\_Required = [yes] AND Moodys\_Rating\_Required = [yes] then the following text is inserted: The Notes have been rated by both S&P and Moody’s  If BOTH Interest\_Basis = [Floating Rate] AND [Interest\_Rate\_Determination] = [Screen Rate] then the following text is inserted: The Interest Rate will be a Floating Rate calculated using a Screen Rate |
| ***Compiled Document***  The Notes have been rated by both S&P and Moody’s  The Interest Rate will be a Floating Rate calculated using a Screen Rate |

### **If Or**

The ‘If Or’ function is another extension of the If function. Similar to If And, If Or tests two sets of arguments and conditions. If either or both tests return true, then the function executes the action. If both tests return false, then the action will not be executed. In If Or functions “OR” appears between the two tests rather than “AND” (as in the If And function).

The conditions and allowed forms of the arguments remain the same as in If And.

|  |
| --- |
| ***GLML Template***  $${If ${S\_And\_P\_Rating\_Required} = [yes] OR ${Moodys\_Rating\_Required} = [yes] ; Insert}{The Notes have been rated.}  $${If ${Interest\_Basis} = [Floating Rate] OR ${Interest\_Basis} = [Fixed to Floating Rate] ; Insert}{The Floating Interest Rate will be determined as described herein.} |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Interest\_Basis | Interest Basis | select\_one |  | | S\_And\_P\_Rating\_Required | Is an S&P rating required? | yes\_no |  | | Moodys\_Rating\_Required | Is a Moody’s rating required? | yes\_no |  |   ***Selectables Table***   |  |  | | --- | --- | | **Name** | **Label** | | Interest\_Basis | Fixed Rate | | Interest\_Basis | Fixed to Floating Rate | | Interest\_Basis | Floating Rate | |
| ***Workflow***  Is an S&P rating required?: [No]  Is a Moody's rating required?: [Yes]  Interest Basis: [Floating Rate] |
| ***System Function Interpretation***  If EITHER S\_And\_P\_Rating\_Required = [yes] OR Moodys\_Rating\_Required = [yes] then the following text is inserted: The Notes have been rated.  If EITHER Interest\_Basis = [Floating Rate] OR Interest\_Basis = [Fixed to Floating Rate] then the following text is inserted: The Floating Interest Rate will be determined as described herein. |
| ***Compiled Document***  The Notes have been rated.  The Floating Interest Rate will be determined as described herein. |

### **Use of Count**

The syntax for calling the Count property of Multi Value tags is described above and it can be used within If functions. The example given shows its use when used within a simple If function.

|  |
| --- |
| ***GLML Template***  $${If ${Supplement Dates; Count} = 1 ; Insert}{  There is one supplement to the Offering Circular.}  $${If ${Supplement\_Dates; Count} > 1 ; Insert}{  There are ${Supplement\_Dates; Count} supplements to the Offering Circular.} |
| ***Data Table***   |  |  |  |  | | --- | --- | --- | --- | | **Key** | **Label** | **Type** | **Value** | | Supplement\_Dates | Supplement Dates | date\_list |  | |
| ***Workflow***  Supplement Dates: [15 March 2020], [15 June 2020] |
| ***System Function Interpretation***  If ${Supplement\_Dates; Count} = 1 then the following text is inserted: There is one supplement to the Offering Circular.  If ${Supplement\_Dates; Count} > 1 then the following text is inserted: There are ${Supplement Dates; Count} supplements to the Offering Circular. |
| ***Compiled Document***  There are two supplements to the Offering Circular. |

# References

[1] Mark-up Language

https://techterms.com/definition/mark-up\_language

[2] Legal XML

http://www2.law.columbia.edu/johnson/lda/readings/SMULRLegalXMLAndStandards.pdf