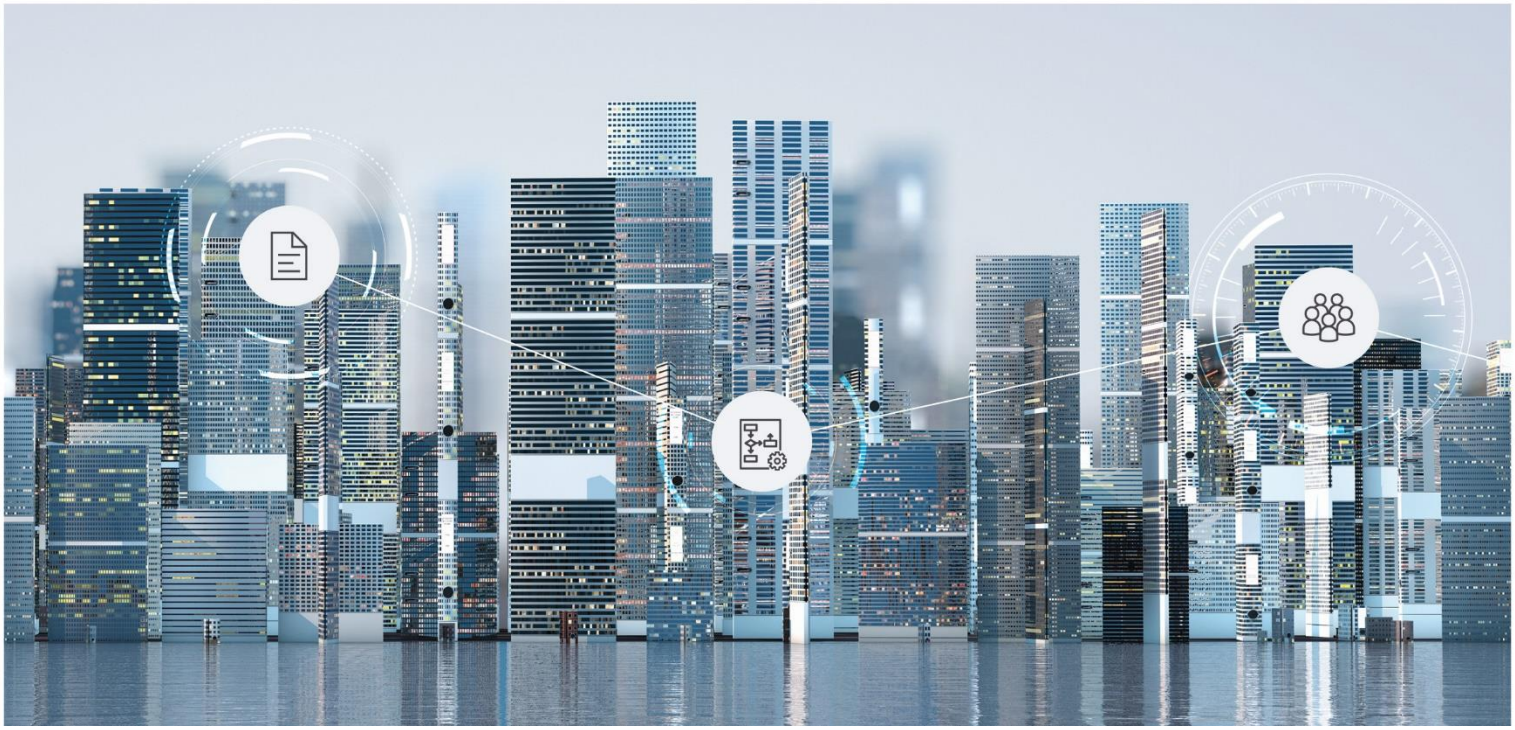
Fabasoft



White Paper

XML-Based Export and Import of Objects Using XSLT

Fabasoft Folio 2023 Update Rollup 2

Contents

1 Introduction

This document describes on the one hand how to export Fabasoft Folio objects to XML data and on the other hand how to create or update Fabasoft Folio objects based on XML data. To accomplish these goals XSLT based actions are provided.

2 Software Requirements

System environment: All information contained in this document implicitly assumes a Microsoft Windows environment or a Linux environment.

Supported platforms: For detailed information on supported operating systems and software see the software product information on the Fabasoft distribution media.

3 Getting Started

All you need to get your first export and import running is contained in this chapter. Detailed information about the introduced approaches can be found in the following chapters. As prerequisite you need a Fabasoft Folio installation with Fabasoft app.ducx. More information about Fabasoft app.ducx can be found in the white paper "An Introduction to Fabasoft app.ducx".

Create a new app.ducx project (in this case `TRANS@1.506`) with a use case and user interface file. Additionally create two XSLT files (`export.xsl` and `import.xsl`) and for the import use case an example XML file (`import.xml`) is required.

Example

app.ducx Use Case Language

```
usecases TRANS@1.506
{
  import COOSYSTEM@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  import COOXML@1.1;
  import GENCONT@1.1;

  // the export should be invoked with a menu command
  menu usecase Export on selected {
    variant Object {
      impl = expression {
        // cooobj contains the object the user has selected
        // in a testing environment it is fast and easy to use files
        // at the end of this chapter an example for working with
        // objects instead is provided
        coouser.XSLTransformObject(cooobj, "c:/trans/export.xsl",
          "c:/trans/result.xml");
      }
    }
  }
  // the import should be invoked with a menu command
  menu usecase Import direct {
    variant Object {
      impl = expression {
        coouser.XSLTransformToObject("c:/trans/import.xml", c:/trans/import.xsl");
      }
    }
  }
}
```

app.ducx User Interface Language

```
userinterface TRANS@1.506
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;

  // the menu commands should be available in the "Tools" menu
  extend menu COODESK@1.1:MenuTools {
    entries = {
      MenuExport,
      MenuImport
    }
  }
}
```

export.xsl

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- the name of the object is exported -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sys="http://www.fabasoft.com/components/COOSYSTEM@1.1">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
   omit-xml-declaration="no" />
  <xsl:template match="/">
    <root>
      <objname>
        <xsl:value-of select="*/sys:objname" />
      </objname>
    </root>
  </xsl:template>
</xsl:stylesheet>
```

import.xsl

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- folders are imported and the name and external key are set -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sys="http://www.fabasoft.com/components/COOSYSTEM@1.1"
  xmlns:desk="http://www.fabasoft.com/components/COODESK@1.1">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
    omit-xml-declaration="no" />
  <xsl:template match="/">
    <xsl:apply-templates select="/root/folder" />
  </xsl:template>
  <xsl:template match="folder">
    <sys:Value search="sys:objexternalkey" searchobjclass="desk:Folder"
     create="desk:Folder" match="literal">
      <sys:objname select="name" />
      <sys:objexternalkey select="externalkey" />
    </sys:Value>
  </xsl:template>
</xsl:stylesheet>
```

import.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- two folders should be imported -->
<root>
  <folder>
    <name>Projects</name>
    <externalkey>projects</externalkey>
  </folder>
  <folder>
    <name>Year 2011</name>
    <externalkey>year2011</externalkey>
  </folder>
</root>
```

If you compile and run this project two menu commands are available at the end of the "Tools" menu. If you did not provide a multilingual name for the menus they will be called "MenuExport"

and “MenuImport”. Put the example files (`export.xsl`, `import.xsl` and `import.xml`) in the defined path on the server running the Fabasoft Folio Web Services. The import command will create two folder objects that can be searched for in Fabasoft Folio. The export command creates an XML file in the defined path.

Working with files is nice for testing but if you do not have access to the file system or you want to set up a productive scenario, here is an extension of the above example:

- Add an object model file to the app.ducx project and define instances for the XSLT files. Put the two XSLT files (`export.xsl` and `import.xsl`) in the resources folder of the app.ducx project.
- Replace the use case file of the previous example with this one.

Example

app.ducx Object Model Language

```
objmodel TRANS@1.506
{
  import COOSYSTEM@1.1;
  import COOXML@1.1;

  // creates a new instance of an XSLT used for export
  instance XSLTransformation ExportXSLTransformation {
    content = file("resources/export.xsl");
  }
  // creates a new instance of an XSLT used for import
  instance XSLTransformation ImportXSLTransformation {
    content = file("resources/import.xsl");
  }
}
```

app.ducx Use Case Language

```
usecases TRANS@1.506
{
  import COOSYSTEM@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  import COOXML@1.1;
  import GENCONT@1.1;

  menu usecase Export on selected {
    variant Object {
      impl = expression {
        // will contain the result XML data
        content @resultxmlcont;
        // the fifth parameter is the out parameter
        // cooobj contains the object the user has selected
        // #ExportXSLTransformation is used as XSLT for the transformation
        @resultxmlcont = coouser.XSLTransformObject(cooobj,
          #ExportXSLTransformation)[5];
        // create a generic content object that will contain the result XML data
        GENCONT@1.1:ContentObject @resultxmlobj =
          #GENCONT@1.1:ContentObject.ObjectCreate()[2];
        @resultxmlobj.content.contcontent = @resultxmlcont;
        @resultxmlobj.content.contextension = "txt";
        @resultxmlobj.objname = "Result XML " + coort.GetCurrentDateTime(cooobj);
      }
    }
  }
  menu usecase Import on selected {
    variant COOSYSTEM@1.1:ContentObject {
      impl = expression {
        // it is assumed that the user selects a content object (cooobj)
        // containingk the source XML data
        // alternatively, you may provide a dialog for uploading an XML file
        coouser.XSLTransformToObject(cooobj, #ImportXSLTransformation);
      }
```

```
    }
  }
}
```

By now you should have your first working example. Detailed explanations of the depicted approaches can be found in the following chapters.

4 Available XLST Based Actions

Fabasoft Folio provides XSLT based actions for import and export. Additionally, an action for standard XSLT processing based on XML input and output is available.

4.1 XSLTransformObject

`XSLTransformObject` is used to export objects to an XML document.

```
COOXML@1.1:XSLTransformObject(
  any[] value,
  any[] xsltfile,
  optional string resultfile,
  optional dictionary parameter,
  optional out content resultcontent,
  optional COOXML@1.1:XMLParser xmlparser)
```

Parameters:

- `value`
  Defines an arbitrary Fabasoft Folio object that should be transformed.
  Example: `coort.GetObject("COO.1.506.1.10126")`
- `xsltfile`
  Defines the XSLT document used for the transformation. It can be a file path, a content or a content object.
  Example:
  `"C:/trans/ExportXSLTransformation.xsl"` (file path)
  `#ExportXSLTransformation.content.contcontent` (content)
  `#ExportXSLTransformation` (content object)
- `resultfile`
  Defines the file path of the result XML file. Alternatively, the result is stored in `resultcontent`, if this parameter is omitted.
- `parameter`
  Defines a dictionary that can contain options for the transformation.
- `resultcontent`
  Contains the result XML data, if the `resultfile` parameter is omitted.
- `xmlparser`
  Defines the XML parser that should be used. Available parsers are: `XMLPARSER_DEFAULT`, `XMLPARSER_MSXML`, `XMLPARSER_LIBXML`.

Note: Keep in mind that the defined file paths belong to the server running the Fabasoft Folio Web Services. File paths may be useful in combination with temporary files. In a development and testing environment (when having access to the file system) it may fasten the development process.

4.2 XSLTransformToObject

`XSLTransformToObject` is used to create or update objects from an XML document.

```
COOXML@1.1:XSLTransformToObject(
  any[] xmlfile,
  any[] xsltfile,
  optional string metaxmlfile,
  optional dictionary parameter,
  optional out content resultcontent,
  optional COOXML@1.1:XMLParser xmlparser)
```

Parameters:

- `xmlfile`
  Defines the source XML document used for the transformation. It can be a file path, a content or a content object.
- `xsltfile`
  Defines the XSLT document used for the transformation. It can be a file path, a content or a content object.
- `metaxmlfile`
  Defines the file path of the meta XML file. Alternatively, the meta XML data is stored in `resultcontent`, if this parameter is omitted.
  The meta XML data is only a by-product of the transformation. Creating or modifying objects in Fabasoft Folio is the main purpose of the XSL transformation and not to transform the source XML to a target document. Thus the target document is called "meta" because it is technically needed for the XSLT process, but it is not the wanted result.
- `parameter`
  Defines a dictionary that can contain options for the transformation.
- `resultcontent`
  Contains the meta XML data, if the `metaxmlfile` parameter is omitted.
- `xmlparser`
  Defines the XML parser that should be used. Available parsers are: `XMLPARSER_DEFAULT`, `XMLPARSER_MSXML`, `XMLPARSER_LIBXML`.

Note:

- `XSLTransformToObject` only provides a limited set of XSLT functionality. Within a property block (e.g. `<sys:objname>My Name</sys:objname>`) no `xsl:if` or `xsl:when` blocks are allowed.
- Keep in mind that the defined file paths belong to the server running the Fabasoft Folio Web Services. File paths may be useful in combination with temporary files. In a development and testing environment (when having access to the file system) it may fasten the development process.

4.3 XSLTransform

`XSLTransform` is used to perform an XSL transformation using a standard XSLT processor. Input and output are based on XML. This action can also be used to pre-process XML input for `XSLTransformToObject` or post-process XML output from `XSLTransformObject`, because these actions only provide a limited set of XSLT functionality.

```
COOXML@1.1:XSLTransform(
  any[] source,
  any[] transform,
  optional out content result,
  optional dictionary parameters,
  optional string filename)
```

Parameters:

- `source`
  Defines the source XML document used for the transformation. It can be a file path, a content or a content object.
- `transform`
  Defines the XSLT document used for the transformation. It can be a file path, a content or a content object.
- `result`
  Contains the XML data, if the `filename` parameter is omitted.
- `parameters`
  Defines a dictionary that can contain options for the transformation.
- `filename`
  Defines the file path of the XML file. Alternatively, the XML data is stored in `result`, if this parameter is omitted.

4.4 Using Parameters

Both `XSLTransformObject` and `XSLTransformToObject` provide the possibility to use a parameters dictionary. The dictionary is available in the global scope when using the `sys:Evaluate` function (see chapter 8.2.4 "sys:Evaluate"). Another way to access the dictionary is to define an `xsl:param` with the same name as the dictionary key.

Example

app.ducx Use Case Language

```
menu usecase Export on selected {
  variant Object {
    impl = expression {
      // dictionary that should be available in the XSLT context
      dictionary @dict;
      @dict.key1 = "valuekey1";
      coouser.XSLTransformObject(cooobj, "c:/trans/export.xsl",
        "c:/trans/result.xml", @dict);
    }
  }
}
```

export.xsl

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:sys="http://www.fabasoft.com/components/COOSYSTEM@1.1">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
   omit-xml-declaration="no" />
  <!-- make the dictionary key available as xsl:param -->
  <xsl:param name="key1" />
  <xsl:template match="/">
    <root>
      <approach1>
        <!-- use xsl:param -->
        <xsl:value-of select="$key1" />
      </approach1>
      <approach2>
        <!-- use sys:Evaluate and access the dictionary in the global scope -->
        <xsl:value-of select="sys:Evaluate('::key1')" />
      </approach2>
    </root>
  </xsl:template>
</xsl:stylesheet>
```

The output looks like this:

Example

result.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<root>
  <approach1>valuekey1</approach1>
  <approach2>valuekey1</approach2>
</root>
```

Note: In complicated cases it may be easier to transport the objects that have to be exported in the parameters dictionary. In this way you can classify and filter the objects beforehand in the app.ducx project and therefore remove complexity from the XSLT document.

5 Working With XSLT

XSLT is used to transform Fabasoft Folio objects to XML data and vice versa. Further on basic knowledge of the XSLT and XPath concept is required. For example, consult http://www.w3schools.com/xsl/ and http://www.w3schools.com/xpath/ for more information about XSLT and XPath.

The libxml2 parser used by Fabasoft Folio includes an implementation for the EXSLT module "Strings" that can be used for easier string manipulation. For more information consult http://www.exslt.org/.

6 Export

6.1 Simple Export Example

The values of an exported Fabasoft Folio object can be accessed via XPath expressions. The following example has already been used in chapter 3 "Getting Started" to export the name of an object.

Example

export.xsl

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- the name of the object is exported -->
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:sys="http://www.fabasoft.com/components/COOSYSTEM@1.1">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
   omit-xml-declaration="no" />
  <xsl:template match="/">
    <root>
      <objname>
        <xsl:value-of select="*/sys:objname" />
      </objname>
    </root>
  </xsl:template>
</xsl:stylesheet>
```

Explanation:

- `xmlns:sys="http://www.fabasoft.com/components/COOSYSTEM@1.1"`
  Defines the `sys` namespace to make component objects from `COOSYSTEM@1.1` accessible with this prefix.

- `<xsl:template match="/">`
  The template matches the root node ("/") that contains the exported object as child.
- `<xsl:value-of select="*/sys:objname" />`
  `*` selects all children of the current node. The current node is the root node (as defined by the `match="/"` statement). The root node has only one child, the exported object. `xsl:value-of` prints out the value of `COOSYSTEM@1.1:objname` of the exported object.

The result XML file looks like this:

Result

result.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<root>
  <objname>My Object</objname>
</root>
```

6.2 Data Types

How to access the data of properties depends on the data types of the properties.

Data Type	XSLT Example	Result Example
string	`<xsl:value-of select="*/sys:objname" />`	my string
integer	`<xsl:value-of select="*/sys:objactversnr" />`	13
float	`<xsl:value-of select="*/term:averagescore" />`	3.49
boolean	`<xsl:value-of select="*/sys:objversnopurge" />`	0
datetime	`<xsl:value-of select="*/sys:objactverscreated" />`	2011-08-10T 12:12:13
object pointer	`<xsl:value-of select="*/sys:objcreatedby/sys:usersurname"/>`	Jones
enumeration	`<xsl:value-of select="*/folio:perssex" />`	2
compound	`<xsl:value-of select="*/sys:objlock/sys:objlocked" />`	0
currency	`<xsl:value-of select="*/my:sum/sys:currsymbol" />` `<xsl:value-of select="*/my:sum/sys:currvalue" />`	4 39000

list data types	`<xsl:for-each select="*/sys:objchildren">`
	`  <xsl:value-of select="sys:objname" />`
	`</xsl:for-each>`
		Pictures
		Documents

Further explanations:

- If the property has no value an empty string is returned. If a part of the XPath has no value, an empty string is returned for the whole XPath.

- Boolean properties return “0” if false, “1” if true and an empty string if undefined.
- Properties of objects referenced in object pointers can be accessed via XPath expressions.
- To each Fabasoft Folio enumeration entry an integer value is assigned. Enumerations return the integer value of the selected enumeration entry.
- Properties of compound types can be accessed via XPath expressions.
- Currency consists of two properties. The property `currvalue` contains the value. The enumeration `currsymbol` contains the currency symbol.
  Note: In some cases it might be useful to replace the integer enumeration value with the enumeration reference (“USD” instead of “0”, or “EUR” instead of “1” and so on). An example on how to achieve this can be found in chapter 8.2.4 “sys:Evaluate”).
- For the data types also corresponding list data types exist. Lists may be looped with `xsl:for-each` or entries can be accessed by specifying the index:
  `<xsl:value-of select="*/sys:objchildren[1]" />` (returns the first entry in the list)

7 Import

7.1 Simple Import Example

Based on a source XML document objects are created or updated. The following example has already been used in chapter 3 “Getting Started” to import two folders.

Example

import.xsl

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- folders are imported and the name and external key are set -->
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:sys="http://www.fabasoft.com/components/COOSYSTEM@1.1"
 xmlns:desk="http://www.fabasoft.com/components/COODESK@1.1">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
    omit-xml-declaration="no"/>
  <xsl:template match="/">
    <xsl:apply-templates select="/root/folder"/>
  </xsl:template>
  <xsl:template match="folder">
    <sys:Value search="sys:objexternalkey" searchobjclass="desk:Folder"
     create="desk:Folder" match="literal" >
      <sys:objname select="name"/>
      <sys:objexternalkey select="externalkey"/>
    </sys:Value>
  </xsl:template>
</xsl:stylesheet>
```

import.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- two folders should be imported -->
<root>
  <folder>
    <name>Projects</name>
    <externalkey>projects</externalkey>
  </folder>
  <folder>
    <name>Year 2011</name>
    <externalkey>year2011</externalkey>
  </folder>
</root>
```

Explanation:

- `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`
  `xmlns:sys="http://www.fabasoft.com/components/COOSYSTEM@1.1"`
  `xmlns:desk="http://www.fabasoft.com/components/COODESK@1.1"`
  Define the needed namespaces. `xsl` and `sys` are mandatory. In this case also the prefix `desk` is defined because the component object `COODESK@1.1:Folder` belongs to `COODESK@1.1`.
- ```
  <xsl:template match="/">
    <xsl:apply-templates select="/root/folder"/>
  </xsl:template>
  ```
  Apply templates to all folder nodes in `import.xml`.
- ```
  <xsl:template match="folder">
    <sys:Value search="sys:objexternalkey" searchobjclass="desk:Folder"
      create="desk:Folder" match="literal" >
      <sys:objname select="name"/>
      <sys:objexternalkey select="externalkey"/>
    </sys:Value>
  </xsl:template>
  ```
  For each folder a corresponding object with the same `objexternalkey` is searched. If no folder is found a new one is created. The properties `objname` and `objexternalkey` are set based on data in the `import.xml`. More information about `sys:Value` can be found in chapter 7.2 "sys:Value".

Note: In most cases it makes sense to work with the property `COOSYSTEM@1.1:objexternalkey` to identify imported and exported objects.

7.2 sys:Value

The `sys:Value` block is used as starting point to redirect the output to a Fabasoft Folio object. Instead of writing to the meta XML file, properties of created or found objects are set.

7.2.1 Searching for Fabasoft Folio Objects

`sys:Value` is used to search for objects in Fabasoft Folio that match the defined criteria. If no object is found a new one is created.

Syntax

```
<sys:Value search="[prop1] [prop2] [...]" searchobjclass="[objclass]"
  create="[objclass]" match="[literal|pattern]">
```

Explanation:

- `search`
  Defines properties to restrict the search. If several properties are defined they are combined with `AND`. The properties used for search have to be defined within the `sys:Value` block (e.g. `<sys:objexternalkey select="externalkey"/>`).
- searchobjclass
  Defines the object class to be searched for.
- `create`
  Defines the object class that is used to create new objects if no object matches the search criteria.
- `match`
  May have two values: `literal` and `pattern`.
  `literal` is translated to `=` in the Fabasoft app.ducx Query Language.
  `pattern` is translated to `LIKE` in the Fabasoft app.ducx Query Language.

Example

```
<sys:Value search="sys:objexternalkey" searchobjclass="desk:Folder"
  create="desk:Folder" match="literal" >
  <sys:objname select="name"/>
  <sys:objexternalkey select="externalkey"/>
</sys:Value>
// the resulting query translated in Fabasoft app.ducx Query Language
SELECT * FROM COODESK@1.1:Folder WHERE .COOSYSTEM@1.1:objexternalkey = 'projects'


<sys:Value search="sys:objexternalkey sys:objname" searchobjclass="desk:Folder"
  create="desk:Folder" match="pattern" >
  <sys:objname select="name"/>
  <sys:objexternalkey select="externalkey"/>
</sys:Value>
// the resulting query translated in Fabasoft app.ducx Query Language
SELECT * FROM COODESK@1.1:Folder WHERE .COOSYSTEM@1.1:objexternalkey LIKE
'projects' AND .COOSYSTEM@1.1:objname LIKE 'Projects'
```

7.2.2 Selecting Fabasoft Folio Objects

Beside searching for objects, objects can also be selected using `sys:Value`.

Syntax

```
<sys:Value select="sys:GetObject('[objaddress]')">
```

Explanation:

- `select`
  Has to return a valid Fabasoft Folio object.

Example

```
<sys:Value select="sys:GetObject('COO.1.506.1.1127')">
  <sys:objname select="name"/>
  <sys:objexternalkey select="externalkey"/>
</sys:Value>
```

7.2.3 Setting Values of Object Properties

There are several types of properties and several ways to set a value.

Note:

- Properties that are updated with invalid values are set empty.
- Properties can be set empty when using empty values in the import XML file (e.g. `<done></done>`).

7.2.3.1 Ways to Set a Value

Static value:

The tag consists of the name of the property to be set.

Example

```
<sys:objname>My Name</sys:objname>
```

Value from an XML document:

The tag consists of the name of the property to be set and a `select` attribute. In the `select` attribute the XPath to value in the source XML is defined.

Example

```
<sys:objname select="name"/>
```

Value evaluated with a Fabasoft app.ducx Expression:

The value is evaluated with the `sys:Evaluate` function. Make sure to use `disable-output-escaping="yes"`, because the output should be as it is without modification.

Example

```
<sys:objname select="sys:Evaluate('&quot;Simple evaluated name&quot;')" disable-
output-escaping="yes"/>

<sys:objname select="sys:Evaluate('::key1')" disable-output-escaping="yes"/>
```

Value evaluated with a Fabasoft app.ducx Expression Using an XML Node:

The value is evaluated with the `sys:Evaluate` function with these parameters:

```
sys:Evaluate('[arbitrary expression]', [arbitrary xpath], ., 'name')"
```

All child nodes of the node defined by the arbitrary XPath are available in the global scope and can be accessed with the node's name. Make sure to use `disable-output-escaping="yes"`, because the output should be as it is without modification.

Example

```
// if the current node is currencysymbol you have access to ::value and ::symbol
// in this case it is used to evaluate the corresponding integer value of EUR
// because the enumeration can only be set with the integer value
<sys:currsymbol select="#CurrencySymbol.typeenumvalues[typeenumref==::symbol].
  typeenumval',./*,.,'name')" disable-output-escaping="yes" />
```

import.xml

```
<currencysymbol>
  <value>203.56</value>
  <symbol>EUR</symbol>
</currencysymbol>
```

7.2.3.2 Data Types

The following example provides an overview on how to set values depending on the data type of the properties.

Example

app.ducx Object Model Language

```
objmodel TRANS@1.506
{
```

```
  import COOSYSTEM@1.1;
  import COOXML@1.1;


  enum EnumTest {
    ET_ENTRY1,
    ET_ENTRY2,
    ET_ENTRY3
  }
  struct CompoundTest {
    string ctprop1;
    string ctprop2;
  }
  // defines a test class with several properties of different data types
  class Test : BasicObject {
    string str;
    integer int;
    float flo;
    boolean boo;
    datetime dt;
    Object op;
    EnumTest et;
    CompoundTest ct;
    currency curr;
    currency currsymb;
    Object[] opl;
  }
}
```

import.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sys="http://www.fabasoft.com/components/COOSYSTEM@1.1"
  xmlns:t="http://www.fabasoft.com/components/TRANS@1.506">
  <xsl:output method="xml" version="1.0" encoding="UTF-8"
    indent="yes" omit-xml-declaration="no" />
  <xsl:template match="/">
    <xsl:apply-templates select="/root/object" />
  </xsl:template>
  <xsl:template match="object">
    <sys:Value search="sys:objexternalkey" searchobjclass="t:Test"
      create="t:Test" match="literal">

      <!-- shows how to set values of properties with different data types -->
      <sys:objname select="name" />
      <sys:objexternalkey select="externalkey" />
      <t:str select="string" />
      <t:int select="integer" />
      <t:flo select="float" />
      <t:boo select="boolean" />
      <t:dt select="datetime" />
      <t:op select="sys:Evaluate('coort.GetObject(::this)', objectpointer)" />
      <t:et select="enumeration" />
      <t:ct>
        <t:ctprop1 select="compound/value1" />
        <t:ctprop2 select="compound/value2" />
      </t:ct>
      <t:curr>
        <sys:currvalue select="currency/value" />
        <sys:currsymbol select="currency/symbol" />
      </t:curr>
      <t:currsymb>
        <sys:currvalue select="currencysymbol/value" />
        <sys:currsymbol select="sys:Evaluate('#CurrencySymbol.typeenumvalues
          [typeenumref==::symbol].typeenumval',./currencysymbol/*,.,'name')"
          disable-output-escaping="yes" />
      </t:currsymb>
```

```
      <xsl:for-each select="list/value">
        <t:opl select="sys:Evaluate('coort.GetObject(::this)', .)" />
      </xsl:for-each>


    </sys:Value>
  </xsl:template>
</xsl:stylesheet>
```

import.xml

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <object>
    <name>Data Types</name>
    <externalkey>datatypes5</externalkey>
    <string>test</string>
    <integer>79</integer>
    <float>50.6</float>
    <boolean>1</boolean>
    <datetime>2012-08-11T10:12:13</datetime>
    <objectpointer>COO.1.506.1.1001242</objectpointer>
    <enumeration>2</enumeration>
    <compound>
      <value1>Project</value1>
      <value2>097845</value2>
    </compound>
    <currency>
      <value>500</value>
      <symbol>1</symbol>
    </currency>
    <currencysymbol>
      <value>203.56</value>
      <symbol>EUR</symbol>
    </currencysymbol>
    <list>
      <value>COO.1.506.1.1001242</value>
      <value>COO.1.506.1.1001241</value>
    </list>
  </object>
</root>
```

8 XSLT Processing

The actions `COOXML@1.1:XSLTransformObject` and `COOXML@1.1:XSLTransformToObject` operate based on a virtual XML source and target document representing object data in Fabasoft Folio. In order to provide this functionality, Fabasoft Folio implements a subset of XSLT that does not cover all aspects of the standard and shows different behavior in some cases. This chapter provides information on limitations as well as new functionality resulting from this.

If more XSLT functionality is required, XML input/output of these actions should be pre-/post-processed using `COOXML@1.1:XSLTransform`.

8.1 XSLT Elements

For object-based transformations, the following elements and attributes are supported:

- `xsl:stylesheet`
- `xsl:include`
  - `href`
- `xsl:template`
  - `match`
  - `mode`

    - name
    - priority
- xsl:value-of
    - disable-output-escaping
    - select
- xsl:apply-templates
    - mode
    - select
- xsl:call-template
    - mode
    - name
- xsl:for-each
    - select
- xsl:output
    - encoding
    - indent
    - omit-xml-declaration
    - standalone
    - version
- xsl:param
    - name
    - select
- xsl:if
    - test
- xsl:choose
- xsl:when
    - test
- xsl:otherwise
- xsl:with-param
    - name
    - select
- xsl:sort
    - datatype
    - order
    - select
- xsl:attribute
    - name
    - namespace
- xsl:element
    - name
    - namespace
- xsl:call-template
    - mode
    - name
- xsl:text
    - disable-output-escaping

- xsl:number
  - format
  - value
- xsl:variable
  - name
  - select

8.2 XPath Evaluation

In the context of COOXML@1.1XSLTransformObject and COOXML@1.1:XSLTransformToObject, Fabasoft Folio evaluates XPath expressions based on object data rather than XML data depending on the XSLT context.

In case of COOXML@1.1XSLTransformObject, XPath expressions specified in XSLT elements are evaluated based on object data. Object properties can be accessed directly using XPath expressions.

Example

```
<xsl:template match="/">
  <name><xsl:value-of select="sys:Value/sys:objname" /></name>
</xsl:template>
```

In case of COOXML@1.1XSLTransformToObject, XPath expressions specified in XSLT elements are evaluated based on XML input data unless elements are generated that represent object data and disable-output-escaping is enabled.

Example

```
<sys:Value ...>
  <sys:objsubject select="name" disable-output-escaping="yes" />
</sys:Value>
```

8.2.1 Limitations

For object-based transformations, XPath expressions evaluated based on XML input and used with xsl:if or xsl:when must return a list of XML elements rather than a Boolean value.

Example

```
<!-- Test for person elements with degree and no publications -->
<xsl:template match="person">
  <!-- NOT OK -->
  <xsl:if test="degree and not(publication)">
  </xsl:if>
  <!-- OK -->
  <xsl:if test="self::node()[degree and not(publication)]">
  </xsl:if>
</xsl:template>
```

8.2.2 Supported Functions

If XPath expressions are evaluated in an object context, the following standard XPATH functions are supported:

- `boolean`
- `ceiling`
- `concat`
- `contains`
- `count`
- `document`
- `false`
- `floor`
- `generate-id`
- `id`
- `lang`
- `last`
- `local-name`
- `name`
- `namespace-uri`
- `normalize-space`
- `not`
- `number`
- `position`
- `round`
- `starts-with`
- `string`
- `string-length`
- `substring`
- `substring-after`
- `substring-before`
- `sum`
- `translate`
- `true`

8.2.3 Additional Functions

Fabasoft Folio provides additional XPath functions that can be used in XSLT elements allowing the specification of an XPath expression (e.g. `xsl:value-of`) in an object context.

Example

```
<xsl:value-of select="sys:GetCurrentUser()" />
<sys:objname select="sys:GetCurrentUser()" disable-output-escaping="yes" />
```

The following XPath functions are available:

- `GetObject(string address)`

- `GetLanguage(string reference)`
- `GetSoftwareProduct(string reference)`
- `GetSoftwareComponent(string reference)`
- `GetObjectClass(string reference)`
- `GetTypeDefinition(string reference)`
- `GetAttributeDefinition(string reference)`
- `GetCurrentDateTime()`
- `GetCurrentDomain()`
- `GetCurrentUser()`
- `GetCurrentUserEnvironment()`
- `GetCurrentUserRoot()`
- `GetCurrentUserLanguage()`
- `GetCurrentUserRoleGroup()`
- `GetCurrentUserRolePosition()`
- `GetCurrentUserRoleSubstUser()`
- `NodeAttributeDefinition(string node)`
- `Evaluate(string expression, optional string globalxpath, optional string localxpath)`
- `HasClass(string objaddress, string objclassaddress)`
- `IsClass(string objaddress, string objclassaddress)`
- `GetReference(optional string objaddress)`
- `GetName(optional string objaddress)`
- `FormatDate(optional date value, optional string formatpattern)`
- `FormatTime(optional time value, optional string formatpattern)`

8.2.4 sys:Evaluate

With the Fabasoft Folio XPath function `sys:Evaluate` it is possible to evaluate Fabasoft app.ducx Expressions. Additionally to the mandatory expression parameter two more parameters (`globalxpath` and `localxpath`) may be defined. The parameters can contain an XPath and the values are accessible within the expression with `::this` and `:>this`.

Example

```
// a simple expression is executed
<xsl:value-of select="sys:Evaluate('coort.GetVersion()')" />
// the optional parameters are used within the expression
<xsl:value-of select="sys:Evaluate('&quot;Name: &quot; + ::this + &quot; Key: &quot; + :>this', */sys:objname, */sys:objexternalkey)" />
// for enums the integer identifier is returned by default
// in this way it is possible to print out the enum reference by using globalxpath
<xsl:value-of select="sys:Evaluate('#CurrencySymbol.typeenumvalues[typeenumval==::this].typeenumref', */my:sum/sys:currsymbol)" />
```

The result may look like this:

Result

```
// version
1127
```

```
// parameters
Name: Projects Key: projects
// currency
USD
```