# IGC Transformer Project.

## Purpose

The main idea to create this transformation was to allow Users to work with Infosphere Glossary Catalog easiest way.

## Terms And Definitions

|  |  |
| --- | --- |
| **Acronym** | **Definition** |
| IGC | Infosphere Glossary Catalog |
| IGCTransformer | Transformation project to transform Input XML file to Output XML file and back |
| Input XML file | File we export from specifically designed EXCEL file |
| Output XML file | File that follows XML schema for import/export IGC assets |
| Input XML schema | Specifically designed schema that represents IGC assets in simple way |
| Output XML Schema | BGSchema.xsd – standard import/export xml files schema for IGC |
| Forward Transformation | Transformation process that passes data from Input XML file to Output XML file |
| Backward Transformation | Transformation process that passes data from Output XML file to Input XML file |
| IGC assets | The subset of objects that Infosphere Glossary Catalog can work with. Category, Term, Label, Custom Attribute, Policy, Rule, Assigned Assets (glossary assets that can be assigned to Term). |

## Overall Process Workflow Description

Workflow of the process contains two main components:

1. EXCEL File that allows End users to work with IGC assets easiest way.
2. IGCTransformer application that transform data between Input XML file and Output XML file.

IGC works with many types of objects (assets). Detailed information about information assets that ICG can work with you can read in the Infosphere Information Server documentation. But key object from Glossary users is Term. According to BGSchema.xsd schema structure of Term element contains complicated hierarchy. Simple child elements of course can be putted easily but when child element contains multichoice structure it is impossible to interpret this structure as tabular data. How to solve this problem? Let us divide Term structure to several parts:

* Simple child element part
* Complex child element part

Simple child element part can be represented by table (list of attributes + list of simple child elements).

For complex child element part let us create set of EXCEL sheet. Complex child elements are Term mappings (set of assets that can be assigned to Term: Logical Attribute/Table, Physical Attribute/Table, physical DB structure table or attribute, Data Stage jobs.). All of these assets have their own attributes so it is impossible to create one common EXCEL sheet.

There are two main flows:

1. Export xml file from IGC.   
   Transform IGC format XML file into EXCEL format xml file.  
   Load EXCEL format xml file into EXCEL template.  
   Use EXCEL file with loaded content to edit, add new assets.
2. Export EXCEL format xml file from EXCEL.  
   Transform exported file to IGC format xml file.  
   Import transformed file into IGC.

## EXCEL template and corresponding xml Schema (xsd) file.

In this we investigate more detailed EXCEL part of the project.   
Start with xml schema file.   
Root element of the schema is “input”

Possible (optional) child elements of the root element can be:

* customAttributesDefinitions – reflect the same element in the IGC format xml
* categories – reflect the same section of the IGC format xml
* terms – reflect the same section of the IGC format xml (except assignedAssets)
* policies – reflect the same section of the IGC format xml
* rules – reflect the same section of the IGC format xml
* labelDefinitions – reflect the same section of the IGC format xml
* termsToSOR – Logical Atomic Model (SoR) mappings (assigned to term assets)
* termsToDM – Logical Data Mart/Interface model mappings
* termsToPhysicalSOR – Physical Atomic Model (SoR) mappings
* termsToPhysicalDM – Physical Data Mart/Interface mappings
* termsToJobs – Data Stage jobs mappings
* termsToImplAssest – RDBMS tables and attributes mappings
* assetReferences – old section not used now.

### Custom attribute definitions section

Let talk more detailed about custom attributes section.

CustomAttributesDefinitions elements can contain zero or more elements “customAttributeDef”.  
customAttributeDef element contains next attributes:

* rid – IGC system internal identifier. String attribute with length of 64 symbols.
* name
* description
* attributeType – string enumeration (TEXT, DATE, BOOLEAN, NUMBER, PREDEFINED\_VALUES, REFERENCE)
* multipleValues – Boolean flag. Indicates that this attribute can have multiple values or not.

There are two more child elements for attribute def:

* validValues – optional element with list of valid values divided by “,” and surrounded by “()”.  
  This is a trick to put attributes with multiple values in the table instead to create one more table.
* appliesTo – optional element with list of values (TERM, CATEGORY, POLICY, RULE) divided by “,” surrounded by “()”

Last two elements have inside one attribute “value” as described above attribute value is a string with multiple values divided by “,” and surrounded by “()”. We will see many attributes like these in the future. In the xml Schema file, I have specifically created Type – “inputValueType” with one attribute “value”.

### Categories section

Let us investigate more detailed categories section.

Optional element “categories” contains zero or more “category” elements.  
Every “category” element contains next attributes:

* rid – IGC system internal identifier.
* name
* shortDescription
* longDescription

and next optional child elements:

* labels – list of label names divided by “,” surrounded by “()”. This element of “inputValueType” type
* stewards – list of user names (IGC steward users) divided by “,” surrounded by “()”. This element also “inputValueType”
* parentCategory – because IGC has categories hierarchy this element identifies parent categories hierarchy. This element of “glossaryAssetRefDef” type and contains two attributes:
  + rid – identifier of parent category (optional if you create new category this attribute must be empty this attribute filled after you export information from IGC).
  + Identity – string attribute reflects categories hierarchy divided by “>>”.  
    Ex. Category1>>Category2>>Category3. Where Category1 is the topmost category and Category3 is the lowest category in hierarchy.

### Terms section

More detailed about terms section.

Terms element contains zero or more “term” child elements.

Every “term” element contains attributes:

* rid
* name
* shortDescription
* longDescription
* status – string enumeration (CANDIDATE, ACCEPTED, STANDARD, DEPRECATED)
* usage
* example
* abbreviation
* additionalAbbreviation
* type – string enumeration (PRIMARY, SECONDARY, NONE)
* isModifier – Boolean flag (TRUE, FALSE)
* Formula – this is an example how to put custom attribute in the list of attributes. Custom attribute starts with Capital letter. Any whitespace in the name replaced by “\_” sumbol.
* Decomposition – another custom attribute
* External\_Document\_Reference - one more custom attribute
* Index – one more custom attribute

And next child elements:

* labels – optional element. list of label names divided by “,” and surrounded by “()”
* stewards – optional element. List of IGC stewards divided by “,” surrounded by “()”
* parentCategory – mandatory attribute. Every term must belong to parent category.  
  the structure of “parentCategory” element described in the categories section.
* isATypeOf – optional element. This element is not used now. But element attribute “value” should contain list of “full term path” values divided by “,” and surrounded by “()”.  
  Full term path – full hierarchy of term categories followed by term name. All values divided by “>>”.

I will come back to terms section when we will talk about terms transformation.

### Policies section

Element “policies” contains zero or more “policy” elements.

Every “policy” element contains next attributes:

* rid
* name
* shortDescription
* longDescription

and next child elements:

* labels - optional
* stewards – optional
* parentPolicy – optional. Element of “glossaryAssetRef” type. Contains two attributes: rid and identity.
* referencedRules – optional element. Element contains list of “full path to rule” values divided by “,” and surrounded by “()”.

### Rules section

Element “rules” contains zero or more “rule” elements.

Every “rule” element contains next attributes:

* rid
* name
* shortDescritpion
* longDescription

and next child elements:

* labels – optional
* stewards – optional
* referencedByPolicies – optional. List of “full path to policy” divided by “,” and surrounded by “()”. Full path to policy contains parent policies hierarchy and policy name. All elements in path divided by “>>”.

### Labels section

Element “labelDefinitions” contains zero or more “labelDefinition” elements.

Every “labelDefinition” element contains next attributes:

* rid
* name
* description

### Mapping sections “termsToSOR”, “termsToDM”, “termsToPhysicalSOR”, “termsToPhysicalDM”

These sections divided to support parallel mapping to Atomic and DM models.

The structure of these section is common. Element “termsToSOR”, “termsToDM”, “termsToPhysicalSOR”, “termsToPhysicalDM” contains zero or more elements “termToSORRef”, “termToDMRef”, “termToPSORRef”, “termToPDMRef”.

Every “Ref” element contains next attributes:

* termCategory – mandatory attribute. Contains categories hierarchy for term divided by “>>”
* termName – mandatory attribute.
* entityName – mandatory attribute. Name of the logical entity in the mapping.
* attributeName – optional attribute. Name of the entity attribute.
* entityPath – mandatory attribute. Full path to the entity (packages hierarchy) divided by “>>”
* namespace – mandatory attribute. When Information Server perform models import every model should contain model namespace. We can have several models imported under the same namespace. And may have the same model imported under different workspaces.

Based on every row in section during transformation we will build Hash Map object.   
Has map is the <key, object> pair. To build key for term we will use “termCategory+termName” (in IGC this is alternative unique key for terms), and collect all “attribute::entity::entityPath::namespace” in the list (we may have more than one mapping for one term).

### Section “termsToJobs”

Element “termsToJobs” contains zero or more child elements “termToJobRef”.

Element “termToJobRef” contains next attributes:

* termCategory – full path to term category divided by “>>”
* termName
* transformServer – name of the DataStage Server
* transformProject – name of the Transformation DataStage project
* jobname – name of the DataStage job

### Section “termsToImplAsset”

The root element “termsToImplAsset” contains zero or more “termToImplAssetRef” elements.

Every “Ref” element contains next attributes:

* termCategory
* termName
* assetHost – Name or IP address of the Database host
* assetDBName – Database name
* assetDBInstance – optional attribute. Database instance
* assetDBMS – optional attribute. Type of RDMBS.
* assetDBSchema – database schema
* assetDBTable – database table name
* assetDBColumn – table column name

## Infosphere Glossary Catalog schema file

This file exported from IGC and contains all possible elements that IGC can operate with.

Root element of the corresponding xml file is “glossary”.

Let me list the sections current project is working with.

### Section “customAttributesDefinitions”

This is optional section and element “customAttributesDefinitions” contains zero or more “customAttributeDef” elements.

Every “customAttributeDef” contains next attributes:

* rid – internal IGC identifier.
* Name – mandatory
* Description
* attributeType – string attribute restricted with list of values (TEXT, DATE, BOOLEAN, NUMBER, PREDEFINED\_VALUES, REFERENCE)
* multipleValues – Boolean attribute indicates that custom attribute may or may not contains multiple values
* inverseReferenceName – optionl attribute This attribute is not used in our scope

and next child elements:

* validValues – optional element. This element contains zero or more “validValue” elements. Each “validValue” element contains two attributes (value – string, description - string). During transformation from EXCEL format file “value” attribute of input “validValues” element split by “,” and for every splitted token “transformer” creates new “validValue” element and put token to the value attribute
* appliesTo – optional element. Element “appliesTo” contains one or many “classType” elements. Element “classType” contains “value” attribute (string type). During transformation I get “value” attribute of the input “appliesTo” element split it with “,” and for every toke create output “classType” element.
* relatesTo – optionl element. Not used in out scope.

### Section “categories”

Element “categories” is optional. Element “categories” contains zero or more “category” element.

Every “category” element contains next attributes:

* rid – system identifier
* name
* shortDescription
* longDescription
* language – not used

and next child elements:

* customAttributes – optional element. This element contains zero or more “customAttributeValue” elements. Each “customAttributeValue” contains two attributes (customAttribute – name of attribute, and value – value of attribute) and two child elements (attributeValue – value of attribute, customAttributeReferences – not used in our scope). Every test export from IGC have not used “value” attribute, all export use child “attributeValue” element. During transformation I collect list of Custom Attributes and create for each “customAttributeValue” element with child “attributeValue” element.
* labels – contains zero or more “label” child element. Every “label” child element contains “name” attribute. During transformation I get input “labels” element and split its “value” attribute by “,”. For every token I create target “label” element.
* stewards – list of steward users. Element “stewards” contains zero or more “steward” element. Every “steward” element contains two attributes (type – always USER, userName). During transformation I get input “stewards” element split “value” attribute by “,” and for every token create target “steward” element.
* parentCategory – full path to parent category. Element “parentCategory” contains two attributes (rid and identity – path to category divided by “::”)

### Section “terms”

Element “terms” contains zero or more “term” elements.

Element “term” contains next attributes:

* rid
* name
* shortDescription
* longDescription
* status – string enumeration (CANDIDATE, ACCEPTED, STANDARD, DEPRECATED)
* usage
* example
* abbreviation
* additionalAbbreviation
* type – string enumeration (PRIMARY, SECONDARY, NONE)
* isModifier – Boolean

during transformation because input attribute names is the same as target I move attributes from source to target as is (with name and value).

Element “term” contains next child elements:

* customAttributes – the same as “customAttributes” in the “categories” section
* labels – the same as in the “categories” section
* stewards – tha same as in the “categories” section
* parentCategory – Mandatory element. Structure is the same as in the “categories” section.
* isATypeOf – rarely used element. Contains zero or more “termRef” elements. Every element contains “identity” attribute is the “full path to term” divided by “::”. During transformation I get “value” attribute of “isTypeOf” input element, split it by “,” and for every token create “termRef” element and put token to “identoty” attribute (replacing input delimiter “>>” to output “::”)
* assignedAssets – most complicated element of this section. This element contains list of term assignments (Logical Entity/Attribute, Physcial Table/Columns, DataStage jobs, Database specifications, tables and columns). To build this target element first of all I build hashtable of all input sections termsToSOR, termsToDm, termsToPSOR, termsToPDM, termsToJobs and termsToImplAssets. Then during working with current term I search through the hashtables for corresponding mappings and depend of the type of asset create corresponding target elements.