

**XML Schema
Flatten & Filter Utility**

**Requirements
Specification**

Date: 5/1/2008 8:53:00 AM

Document Contacts:
Steffen M. Fohn
Isabel Espina
ADP

Document Version: 1.4.1

Table of Contents

Table of Contents	2
Document Control	3
Version History	3
Overview.....	4
Purpose	4
Scope	4
Use Case Model	5
Use Case: Flatten & Filter a Schema File Set.....	5
Basic Flow	5
Associated Nonfunctional Requirements	6
Exception Flow.....	8
Exception Flow 1 – System cannot process the request.....	8
Associated Nonfunctional Requirements	8
Glossary	10
Domain Model	11
Generalized Model.....	11
OAGIS 9.0 Library Model	12
Other Library built on OAGIS 9.0.....	13
References.....	13
Appendix.....	13
OAGi Requirements Input	13
Re: Michael’s review of the requirement specification v1.1 on 04/21/05	13
Inputs.....	14
Processing.....	14
Outputs	14
Re: Exchange of Questions/Answers (Michael and Steffen 04/13/05-04/22/05).....	15
Re: Statement of Flattener Intent (Michael on 04/13/05)	15

Document Control

Version History

Date	Updated By	Version	Change Description
04/11/05	Steffen Fohn	1.0	Initial Version
04/15/05	Steffen Fohn	1.1	Updates: <ol style="list-style-type: none">1. Updated the Overview section2. Generalized the terminology of the requirement to apply to any schema (xsd) file.3. Added non-functional requirements, #3 and #4
04/25/05	Steffen Fohn Michael Rowell	1.2	This version contains updates from Michael Rowell's Review of Version 1.1. His original comments are located in the Appendix. Updates: <ol style="list-style-type: none">1. Broadened the Use Case to represent a Schema File Set (one or more schema files) instead of only one Schema file.2. Organized and updated the non-functional requirements. Non functionals #1, 2, 3, 7, and 8 were added; #4 was updated.3. Added an Appendix that documents correspondence with OAGi on the Flattener.
05/10/05	Steffen Fohn	1.3	Update Figures 2 and 3 were update to reflect changes found in the file organization between OAGIS 9.0 Beta and GA versions.
03/26/07	Isabel Espina	1.4	Extend Use Case to support Filtering and Flattening of a set of related schema definition files (xsd).
04/25/07	Steffen Fohn	1.4.1	Updated document to reflect internal review of the previous version: <ol style="list-style-type: none">1. Basic Flow Use Case steps 1, 6, and 6a were updated2. Nonfunctional requirements were categorized/organized3. Nonfunctional requirement 3b was updated.

Overview

Purpose

The purpose of this work is to build a utility that is capable of “flattening” and “filtering” an XML Schema Definition (XSD) file, or a collection of related XML Schema Definitions (XSD) files. The XML Schema (XSD) file is referred to as a *schema file* throughout this document. The collection of related XML Schema Definition (XSD) files is referred to as a *schema package* throughout this document. A *dependent schema file*, by virtue of being *dependent*, references other schema files through “include” and “import” statements. The “include” is used by a dependent schema file to reference other schema files, in the same namespace, that contain schema content (i.e., complexType, simpleTypes, groups and elements) needed or used by the dependent schema file. The “import” is used by a dependent schema file to reference other schema files, in different namespaces, that contain schema content needed by or used by the dependent schema file. Note that an “included” or “imported” schema file may also be a dependent schema file, having its own “include” and “import” statements; this nesting of file references repeats until the “included” or “imported” file no longer reference another schema files.

The schema package is a collection of Schema Definitions which are related in some context known by the user. An example of a context may be that all the schema files map to one interface.

From a schema library development perspective it is desirable to minimize the number of schema files to simplify management of the “include” and “import” statements. However, from a code development, test and operational architecture perspective, this type of file structuring/organization has the undesirable effect of “including” and “importing” schema content that is not relevant to the XML schema at hand (the dependent schema file). This is because the “include” and “imports” statements are at a file level or scope. As such, a dependent schema “includes” and “imports” *all* of the schema content contained in the “included” or “imported” schema files. This is compounded for the schema package since it is comprised of dependent schemas, which results in the duplication of all relevant and irrelevant components. Changing the development library file structuring/organization to avoid this effect is not a viable option as it would complicate schema development.

Specifically, the following qualities are required for an XML schema or XML schema package to support code development, test, and operational architecture needs.

- contains/references schema content that is relevant to the schema file (the dependent schema file) or package (set of dependent schema files)
 - this allows for generation of documentation of relative schema content
 - this simplifies communication with stakeholders
 - this increases performance of schema validation and data binding
- minimizes the number of schema files simplifying schema deployment

Recognizing the needs of schema and code development, test and operational architecture, a utility is required that recursively processes a dependent schema file or a package (set of dependent schema files) and its referenced schema files that may exist across different namespaces in a schema development library and outputs a minimal set of schema files (one file for each namespace) with schema content that is relevant to the dependent schema or schema package respectively.

Scope

The scope of this work is described through the requirements in the following sections. This consists of a single use case, associated non-functional requirements, and a domain model

Use Case Model



Use Case: Flatten & Filter a Schema File Set

Basic Flow

Name	Flatten and Filter a Schema File Set
Preconditions:	<ol style="list-style-type: none"> 1. The input schema files (i.e., BODs) are valid (well-formed). 2. The input schema files (i.e., BODs) and its constituent schema files (i.e., imported and included schema files) are available in the schema development library (i.e., BOD development library).
Actor:	Schema Developer

1. The Schema Developer starts the flattening and filtering utility.	The system offers the choice of flattening and filtering: <ol style="list-style-type: none"> 1. specific schema files “Schema File” (where each file is flattened and filtered distinctly) <ol style="list-style-type: none"> a. one or more schema files may be selected b. all schemas files located in a specified directory 2. a package of schema files “Schema File Package” (where the set of schema files will be filtered and flattened collectively/together)
1a. IF the Schema Developer selects a “Schema File Package” THEN the Schema Developer specifies a name for the package.	The system prompts for and stores the package name.
3. The Schema Developer navigates the schema development library (i.e., OAGIS library).	The system supports requests for file directory navigation and returns directory file contents.
4. The Schema Developer selects schema file(s) or a directory that contains one or more schema files (i.e., BODs) as input for the flattening and filtering process.	The system accepts and confirms either the input schema file(s) or the input directory as specified by the Schema Developer.
5. The Schema Developer specifies a target directory where the flattening and filtering process will save the output directories and schema file(s).	The system accepts and confirms the directory location for the output directories and schema files.
6. The Schema Developer requests the system to process the input schema file(s) (i.e., BOD).	The system starts processes the flattening and filtering request.

6a. IF the Schema Developer had chosen to flatten a schema file package and { IF the schema input files cross multiple namespaces, THEN the Schema Developer specifies a namespace to be associated with the “root” or package-level output file.	The system queries the Schema Developer to select the namespace to be associated with the “root” or package-level output file
7.	The System generates the schema output files.
Postconditions:	The flattened and filtered output directories and schema file(s) are available in the specified target directory.

Associated Nonfunctional Requirements

1. Naming the output directory

If a package is requested then the output directory will have the same name as the package. Directory structure may be represented as <targetdirectory>/<packagename>. **Otherwise**, the system will create an output directory with the same name as the input schema file.

For example:

“Schema File Package” option:

Schema Files selected for flattening and filtering:

ProcessPurchaseOrder.xsd

AcknowledgePurchaseOrder.xsd

ConfirmBOD.xsd

Package Name is PurchaseOrder

Output Directory → <targetdirectory>/PurchaseOrder

“Schema File” option:

Schema Files selected for flattening and filtering:

ProcessPurchaseOrder.xsd

AcknowledgePurchaseOrder.xsd

Output Directory → <targetdirectory>/ProcessPurchaseOrder

Output Directory → <targetdirectory>/AcknowledgePurchaseOrder

Note: <**output directory**> will be used to reference where the output schema files and subdirectories will be created. This will differ based on whether the Schema Developer has requested to flatten and filter as a package or to flatten and filter individual schema files.

2. Naming the output files of imported schema files

For each *imported* file of a single namespace used by an input schema file, the system will create an output subdirectory with the same name as the namespace. Continuing with the Schema File example, above, assume that both ProcessPurchaseOrder.xsd and AcknowledgePurchaseOrder.xsd are defined in the <http://www.openapplications.org/oagis/9> namespace and import content from the <http://www.other.com> namespace. The target and output subdirectory structure for the input schema file, ProcessPurchaseOrder would be <targetdirectory>/ProcessPurchaseOrder/www.other.com. The target and output directory structure for the input schema file, AcknowledgePurchaseOrder would be <targetdirectory>/AcknowledgePurchaseOrder/www.other.com. In summary, the directory structure is represented as <**output directory**>/<namespace>.

The Schema File Package option would follow the same approach.

3. Naming the output file of the root schema file

- a. If the Schema File option is chosen, then the input schema file name, defined on a given namespace, will be used as the name of the output file for that namespace and stored in the **<output directory >** directory. For example, the ProcessPurchaseOrder.xsd is defined in the <http://www.openapplications.org/oagis/9> namespace and has a name “ProcessPurchaseOrder”. The system will create an output schema file named ProcessPurchaseOrder that includes all the *needed or used* schema content for that namespace. The system will store the file in the **<output directory >/<ProcessPurchaseOrder>** directory.

- b. If the Schema File Package option is chosen, then package name will be used as the name of the output file for that namespace and stored in the **<output directory >** directory. For example,

Package Name = PurchaseOrder

Schema Files selected as part of the package and their corresponding namespace:

ProcessPurchaseOrder.xsd -> <http://www.openapplications.org/oagis/9>

AcknowledgePurchaseOrder.xsd -> <http://www.openapplications.org/oagis/9>

ConfirmBOD.xsd -> <http://www.other.com>

The system will create an output schema file named PurchaseOrder (corresponding to the package name) that includes all of the *needed or used* schema content for that namespace. The system will store the file in the **<output directory >/<PurchaseOrder>** directory. The ConfirmBOD content will be included under the <http://www.other.com> subdirectory as per requirement 5 below.

In the Schema File Package Use Case, it is important to note that the input schema files may exist across different namespaces. This is illustrated in the example, above, where the ProcessPurchaseOrder.xsd and AcknowledgePurchaseOrder.xsd reside in one namespace and the ConfirmBOD.xsd resides in a different namespace. This “begs the question”...”which namespace should be represented by the root or “package-level” output schema file?” The namespace must be specified the schema developer.

4. As stated in above, when an “imported” namespace is encountered during the flattening and filtering process an output schema file shall be created with the same name as the “imported” file name. All of its “included” schema content that is needed by or used by the input schema must be stored in this file. The system will store the file in the **<output directory >/<namespace>** directory. For example, the ProcessPurchaseOrder imports schema content from an imported file called OtherComponents.xsd defined in the <http://www.other.com> namespace. The system will create an output schema file named OtherComponents.xsd that includes all of the needed or used schema content for that namespace. The system will store the file in the **<targetdirectory >/<ProcessPurchaseOrder >/<www.other.com >** directory. Note that only one output schema file will be created for each namespace used by the input schema. Specifically, one output file for the namespace of the input schema and all of it “includes” and one output file for each “imported” namespace. The output file created for a given namespace should represent the “highest-level” file in a schema file hierarchy (see glossary).
5. The Schema Flattening and Filtering Utility will use the schemaLocation attribute on the xsd:import and xsd:include elements to resolve the location of the BOD schema’s constituent schemas (and their constituent schemas, ad infinitum).
6. The Schema Flattening and Filtering Utility will only accept schema files as input for flattening and filtering.

7. All forward slashes, “/”, used in a namespace shall be translated to underscores “_” when the namespace is used as a directory name. For example, www.other.com/v1 is translated to www.other.com_v1 as a directory name.
8. In the creation of schema output files only the used or needed content should be included.
 - a. All files encountered through the include statements should be searched for content that the source schema file uses as well as content that the new content that is used uses. In other words if a type in the BOD file uses a type in an included file and that type references an element. The original type, the included type, the referenced element and everything that elements uses should be copied to the flattened files. (The existing code does this.)
 - b. All files encountered through the import statements should be searched for content that the source schema file uses as well as content that the new content that is used uses. In other words if a type in the BOD file uses a type in an imported file and that type references an element. The original type would be moved to the original namespaces target file. The imported type, the referenced element and everything that elements uses assuming they are all in one namespace should be copied to a file named the same as the imported file.
9. **If** the package option is chosen the schema output files include the used or needed content -from all the chosen schemas following requirement 9a and 9b above.

Exception Flow

Exception Flow 1 – System cannot process the request

Name	System cannot process request
Preconditions:	<ol style="list-style-type: none"> 1. The input schema files (i.e., BODs) are valid (well-formed). 2. The input schema files (i.e., BODs) and its constituent schema files (i.e., imported and included schema files) are available in the schema development library (i.e., BOD development library).
Actor:	Schema Developer

1. The system will issue an error message when the requested process cannot be completed by the system for any reason (for example, circular reference detected in schema).	The system supports the handling and displaying of error messages for debug purposes.
---	---

Postconditions:	The system will return to the state prior to the inception of the use case.
-----------------	---

Associated Nonfunctional Requirements

1. The Schema Flattening and Filtering Utility will write error messages to a log file. The log file maybe referenced as follows: **<output directory>/log/error.log** The Schema Flattening and Filtering Utility will abort *if* it encounters errors while flattening and filtering a set of schemas

Glossary

BOD – OAGi Business Document Object

Constituent Schema File – a schema file that is “included” or “imported” by a different schema file

Input Schema File – a schema file that is input into the flattening and filtering process. The input schema file is referred to as a dependent schema in the overview, above.

Schema Package – A set of schema definitions (xsd) related for a specific purpose (e.g. A set of BODs referenced by an interface).

Flattening – is the process traverses the schema file hierarchy from the root node through the descendent node’s. For each descendent node (schema file), its file’s schema content (i.e., complexType, simpleTypes, groups and elements) is written into a single file. One can perceive this activity as compressing or flattening the hierarchy into a single file.

Filtering – is a process that occurs during flattening. Flattening does not prevent the inclusion of schema content (i.e., complex and element types) that is not needed/used by the elements types designated in the root node (schema file). Consider an example using the OAGi BOD Architecture, a specific BOD.xsd includes a specific Noun.xsd which includes the Components.xsd. The Components.xsd contains all of OAGi components; only a subset of these components are actually needed/used by the element types for the specific BOD.xsd. Consequently, those components not being used may be filtered from a “flattened” file. This filtering concept applies to all the descendent nodes (schema files) of the Schema File Hierarchy.

Output Schema File – a schema file that is created as an output of the flattening and filtering process.

Schema Content – schema content refers to the complex and element types defined in a schema file.

Schema File – is an XSD (XML Schema Definition) file

Schema File Set – is a set of schema files (one or more)

Schema File Hierarchy – a schema file may “include” and/or “import” one or more constituents schema files; that constituent file may “include” and/or “import” other constituent schema files. The recursive “include”/“import” structure manifests itself as an n-level hierarchy of nodes where each node represents a schema file and each link between nodes represents either an “include” or “import” association. The file designated for flattening is always considered to be the root or top node of the hierarchy.

Input Directory – directory specified by the user that contains the input schema files

Output Directory – is always a subdirectory of the target directory that is created as an output of the flattening and filtering process

Target Directory – directory specified by the user that will contain the output directories and schema files

Domain Model

The Generalized model should serve as the domain model for this work. The OAGIS 9.0 and Other Library model are provided to illustrate examples of development libraries and their file structure.

Generalized Model

Figure 1 represents the most general form the domain model is represented as a hierarchy of nodes where nodes represent the schema files and the “import” and “include” statements in a schema file represent the links between the nodes.

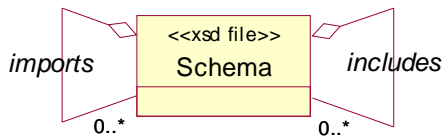


Figure 1

Figure 2 represents the schema “package” which is realized in an xsd file. A package includes 2 or more Schema File Sets.

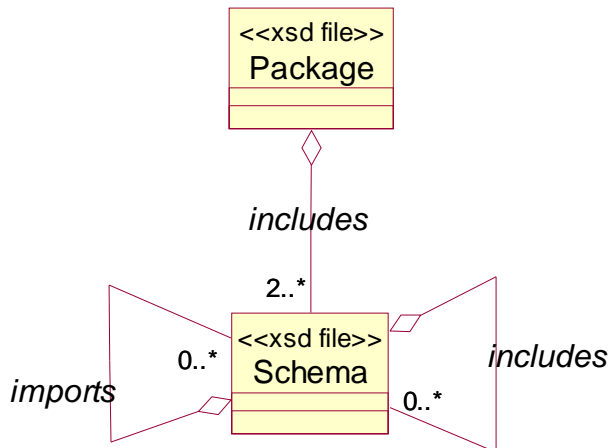


Figure 2

OAGIS 9.0 Library Model

Figure 2 shows the schema file structure in place for OAGIS 9.0

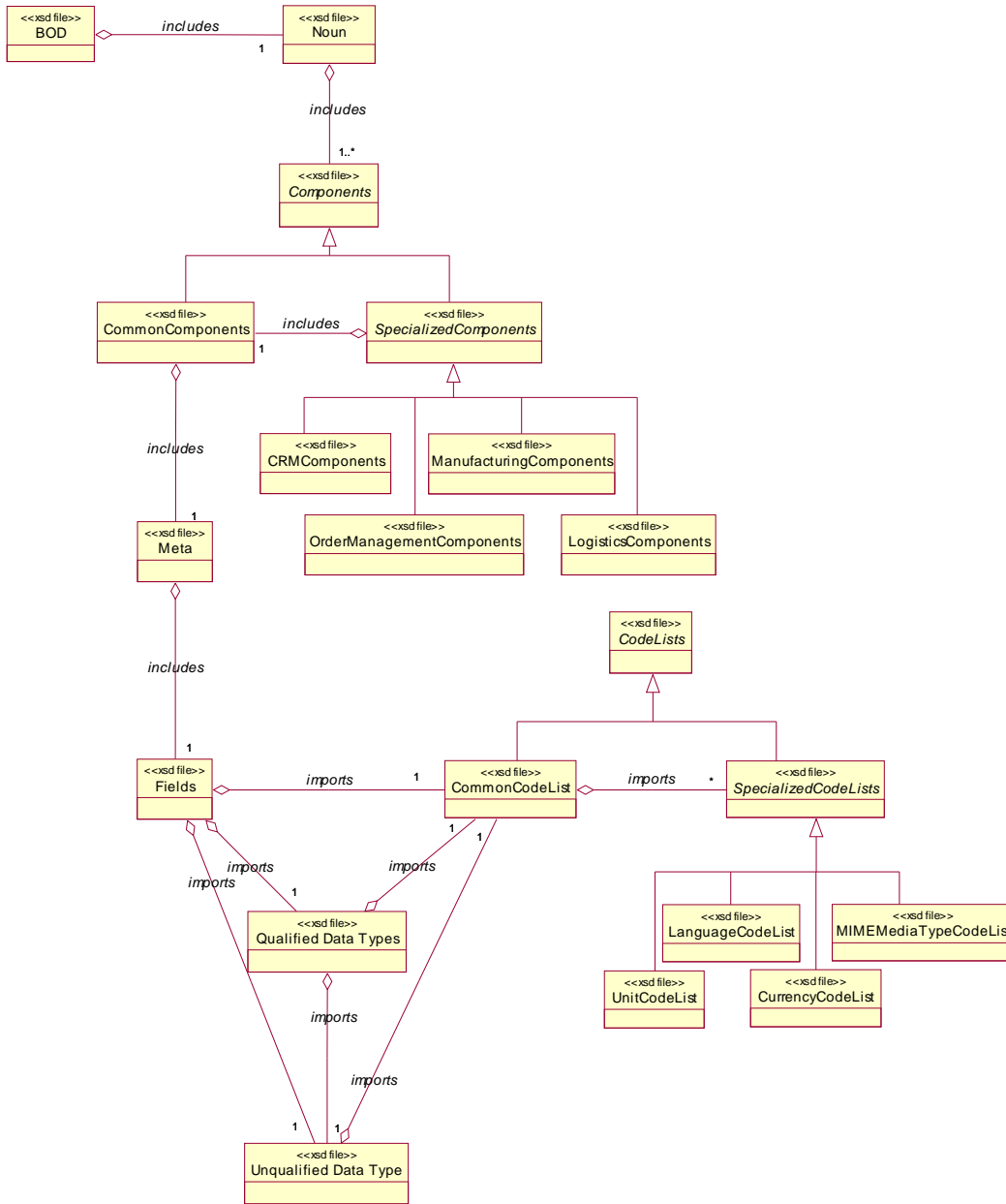


Figure 2

Other Library built on OAGIS 9.0

Figure 3 shows a sample schema file structure for some “other” namespace that has built its library using the OAGIS 9.0 library.

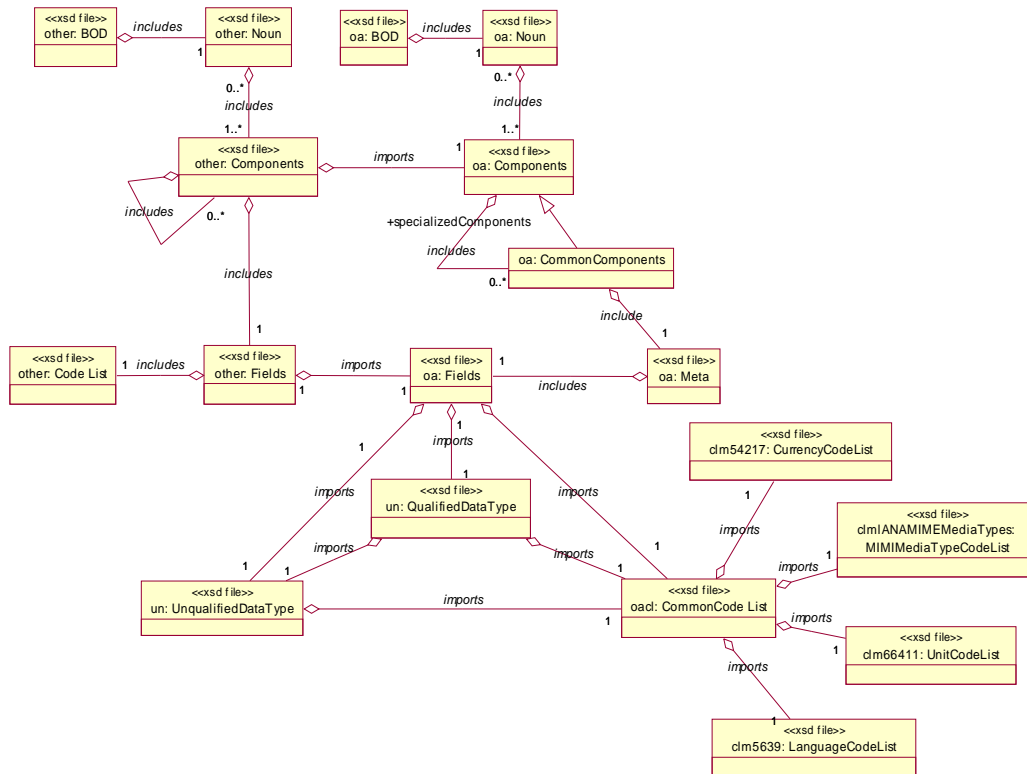


Figure 3

References

OAGi, OAGIS 9.0 Library

Appendix

OAGi Requirements Input

Re: Michael’s review of the requirement specification v1.1 on 04/21/05

The following is how the flattener should work via a graphical GUI:

Inputs

- An input file or directory - In the case that the input is a directory the intention is to repeat the process for each *.xsd file in the directory. Placing a file named the same as the file in the source directory to the target directory.
 - In OAGIS 9.0 the number of BODs are 434 without the capability of processing a directory the flattener would have to be 434 times.
- A target directory – Is the location in which the flattened BODs are to be placed.

Processing

- Since the nature of the “includes” and “imports” in schema are recursive in nature i.e. there maybe any number of includes and any number of imports to any number of levels, it is recommended to do this in a recursive manner as the code provided does.
- The purpose of this processing is to identify the content (complexType, simpleTypes, groups, Elements) that are being used.
 - Elements may be referenced through the ref attribute
 - simpleTypes maybe used by the restriction base attribute, the union membersType (where they are listed separated by a whitespace), the extension base attribute.
 - complexTypes maybe used by the extension base attribute.
 - For others please refer to the code provided as it does this presently for the include side. Basically the import side needs to work in a similar way except the results from an give import namespace should be placed in a separate file.
- Only the used content should be moved to the Flattened files.
 - All files encountered through the include statements should be searched for content that the source schema file uses as well as content that the new content that is used uses. In other words if a type in the BOD file uses a type in an included file and that type references an element. The original type, the included type, the referenced element and everything that elements uses should be copied to the flattened files. (The code provided does this today.)
 - All files encountered through the import statements should be searched for content that the source schema file uses as well as content that the new content that is used uses. In other words if a type in the BOD file uses a type in an imported file and that type references an element. The original type would be moved to the original namespaces target file. The imported type, the referenced element and everything that elements uses assuming they are all in one namespace should be copied to a file named the same as the imported file. (Please see the Outputs section below for the location of these import files.)
 - There will be a need to manage all of the content used from a in a given namespace in a the highest level of the files imported....For example, if OAGIS 9.0 ProcessPurchaseOrder is imported into a 2nd namespace in a file ProcessPurchaseOrder and in another file say PurchaseOrder in the same 2nd namespace the OAGIS 9.0 PurchaseOrder schema is imported the content used should be placed in the ProcessPurchaseOrder file for the OAGIS 9.0 namespace.

Outputs

Since the content found to be used from a different namespace for a given BOD will most likely differ from other BODs the results will need to be kept separated. The current version of the flattener does this by placing the content into a file named the same as the original in the targetdirectory. However, for imported namespaces this would not be enough as many times the same namespace and files can be imported with differing usage. The results must be separated further based on the BOD in which they are to be used.

My recommendation for doing this would be to place the content of the flattened BOD in an additional subdirectory of the targetdirectory i.e., If the BOD ProcessPurchaseOrder is being flattened the resulting schema for the original file would be created at
<targetDirectory>/ProcessPurchaseOrder/ProcessPurchaseOrder.xsd.

All content of namespaces used by ProcessPurchaseOrder would be placed in an additional subdirectory named after the namespace in a file of the same name of the imported file. For example: The OAGIS Codelist file has the following namespace <http://www.openapplications.org/oagis/9/codelists> , different from OAGIS 9.0 namespace. Each OAGIS 9.0 BOD uses this additional namespace. The location of this CodeList file should be:

<targetDirectory>/ProcessPurchaseOrder/www.openapplications.org_oagis_9_codelists/CodeList.xsd

Re: Exchange of Questions/Answers (Michael and Steffen 04/13/05-04/22/05)

1. Specifying a source directory (or rather one or more input files)...is the rationale here to build one file (for each namespace imported that includes **all** the content used by the **multiple** input files? I see this as an additional use case that would need to be added to the spec. From a development iteration perspective, can we put this use case of multiple input files in a subsequent next iteration?

[Michael Rowell] The reason for taking a directory is so that when the pointed at a directory all of the files (BODs) in that directory would be flattened one file per file in the target directory... In other words the source and target directories should have the same number of files in them. The only difference is the target files will contain everything each needs...

[Fohn] Seems straightforward for an input file defined over single namespace...but seems to be more complicated if the input file is defined over imported namespaces; if one file is output for each imported namespace, in order to keep output files generated from imported namespaces from being overwritten each input file would require its own target directory.

<mlr>

Yes, see the Functional Requirements section in the document in the earlier email for more details of this....the output file will need to be placed in a directory like <targetDirectory>/ProcessPurchaseOrder/ProcessPurchaseOrder.xsd – where processing the ProcessPurchaseOrder file. Additionally, the imported namespaces may have a file of the same name so, they should be placed in the location

<targetdirectory>/<BODDirectory>/<namespace>/file.xsd ie.

<targetdirectory>/ProcessPurchaseOrder/www.openapplications.org_oagis_9_codelist/Code Lists.xsd

,/mlr>

2. Version as a parameter...is this an input parameter? Not sure as to why version is needed as a parameter since we've generalized the problem/solution to any xml schema and since the version information is designated in the namespace.

[Michael Rowell] This is not needed for 9.0... it is a left over from OAGIS 8.0 where the version number was not part of the namespace...and not necessarily part of the schemaLocation. You need to make sure that the code handles both options as generically the version may not be a part of the namespace

[Fohn] Where is version information stored in OAGIS 8.0?

<mlr>

Its an attribute on the XML Instance and the schemaLocation path but not in the namespace.

</mlr>

Re: Statement of Flattener Intent (Michael on 04/13/05)

The intent for the flattener is as follows: (The flattener is written in recursively in that there really is no limit to the number of includes or how deep those includes may go... The same approach has to be used for the import side.)

1. Parameters: <source> a directory or file for the source, (where if it is a directory process all the files in that directory). <target> A target directory where to store the resulting schemas. <version> The version of OAGIS being flattened. Another parameter not currently in the source that will be needed is the name of the Overlay and version if applicable. Another parameter that will be needed is the root element of the schema (this will need to be a list of root elements and types that we are using in this namespace for later)...for the BODs this can be assumed for
2. The base target file that is produced has the same target namespace as the source file provided. Where it goes across all of the includes finding everything the root element uses in this namespace...we simply need to add the import side.
3. When an import is encountered the flattener should create a file with the same name as the imported file in a directory named the same as the namespace of that file. This file will have to include everything that the original source file uses from this namespace. Basically this is a repeat of step 2 and then 3... where we will need to find a list of elements and types that we need to have from the imported namespace...
4. Okay the above will work for us if we are importing a single file from a given namespace...If there are multiple files being imported from the same namespace we need to realize that we may not find a given element or type in one namespace diveso we need to come back with a list of elements/types that we did not find so that when the next file import from that namespace is encountered we can use this list for that files hunt...

If the original file is valid then we should find every element or type used by the given element. If not the file may not be valid originally.