Using IMS Content Packaging to Package Instances of LIP and Other IMS Specifications

Version 1.0 Implementation Handbook

Document Name: Using IMS Content Packaging to Package Instances of LIP and Other IMS Specifications
Revision: 9 August 2001

Table of Contents

1. Introduction

This IMS Implementation Handbook addresses the use of the IMS Content Packaging specification for packaging multiple instances of Learner Information Package specification, together with any associated portfolio files, into a single container or Package for transport between systems. The method outlined may also be used for packaging other documents produced in conformance with any other IMS specifications.

This method does not require any changes to the existing Content Packaging specification and hence has no impact on its original use for packaging content.

A Learner Information Package instance (LIP-i) is an instance of the IMS Learner Information XML binding. It is assumed that the information used to construct a LIP-i is normally held in a database.

An IMS LIP-i may have portfolio files or certificate images associated with it. These files may be either referenced externally with a URL, or they may be locally referenced files, in which case these files may need to be packaged along with the Learner Information File. This guide shows how to associate an IMS LIP-i with its particular set of files when these are to be included in the larger set of files contained in a Package. It also recognizes that it is possible for different files associated with different LIP-is to have the same name (e.g., myphoto.jpg). A best practice mechanism is provided for resolving potential file name clashes. This mechanism requires support for a structured directory in the packaging container.

A general method will be presented that supports the compound aggregation of one or more Learner Information Package instance files where each might reference further files (e.g., portfolio files, etc.) which are to be included in the Package. For those situations in which the LIP-is have no attached files, a simplified method is also presented.

In this Handbook, it is recommended that a separate LIP-i is created for each learner, rather than use the facility in LIP to package multiple learner profiles in a single LIP document. Therefore a LIP-i is taken to hold information about a single learner. It is assumed that LIP-is are independent of each other. However, it is suggested that the Content Packaging's <organizations> may be used to group and structure LIP-is, should this be desired.

Note: Where multiple learner information sets are to be packaged, but they have no associated files that need to be included, then the existing facility within LIP to package multiple learner information sets into a single LIP instance could be used as an alternative to the simplified method. The resulting file could be included in a Content Package referenced by a single <resource> element, perhaps with instances of other IMS specifications. However this option is not explored further in this Handbook.

2. General Packaging IMS Learner Information Instances

The general method handles LIP-is that have individually associated referenced files such as portfolio components and images. These referenced files may themselves be contained in a directory structure, and may have relative cross-references to each other that require that this directory structure is maintained. The packaging method described here creates a “compound” structure that contains both an IMS XML instance file and any resource files that it references. In outline form, the steps are as follows:

1) Create a directory in local storage for the Package as a whole.

2) Within this directory, for each Learner Information record to be packaged, create a sub-directory, with a locally unique name, which will hold the Learner Information Package instance, together with its associated files.

3) For each Learner Information instance:
(i) place in the sub-directory all files referenced by a Learner Information instance. Include any further files that they in turn reference with relative addresses. Maintain their sub-directory structure as necessary.
(ii) create the IMS Learner Information File, adding into its relevant sections references to all included files, using a relative address that assumes the Learner Information instance file’s assigned sub-directory as its root.
(iii) place the Learner Information Package instance as a file in the sub-directory.

4) Build the Manifest for the Package and add it to the top-level directory for the Package as a whole.

5) Create the Package by copying the entire contents of the Package directory, including all its sub-directories, into the transfer container.

The following is an example of the contents of a Package root directory and its file directory structure:

```
imsmanifest.xml

LIP1/LIP.xml
LIP1/MyPix.jpg
LIP1/Transcript.pdf
LIP1/Logo.jpg
LIP1/Project1.html
LIP1/Project1_images/tree.gif
LIP1/Project1_images/bush.gif
LIP1/Project1_images/shrub.gif

LIP2/LIP.xml
LIP2/MyPhoto.jpg
LIP2/Transcript.pdf
LIP2/Logo.jpg
LIP2/ReportIndex.html
LIP2/Reports/Project1.html
LIP2/Reports/Project1/Project1_images/pebble.gif
LIP2/Reports/Project1/Project1_images/rock.gif
LIP2/Reports/Project1/Project1_images/stone.gif
LIP2/Reports/Project2.html
LIP2/Reports/Project2/Project2_images/man.gif
LIP2/Reports/Project2/Project2_images/woman.gif
LIP2/Reports/Project2/Project2_images/baby.gif

LIP3/LIP.xml
LIP3/MyPix.jpg
LIP3/UMTranscript.pdf
LIP3/Logo.jpg
LIP3/Project1.html
LIP3/Project1_images/tree.gif
LIP3/Project1_images/bush.gif
LIP3/Project1_images/shrub.gif
  . . .
```

2.1 Create a Directory in Local Storage for the Package as a Whole

2.2 Create a Sub-Directory for the Learner Information Instance

Create a new sub-directory with a name that is unique within the Package for each Learner Information Package instance that is to be packaged (e.g., LIP1, LIP2, LIP3).

2.3 Add Each Learner's Information Files

Portfolio files associated with a LIP-i are likely to be stored at the source site in locations determined by local conventions. Portfolio references may only have local significance and are not necessarily to be replicated at the destination site where different conventions and storage locations may apply. However, all relative addressing between files needs to be maintained within a Package, to avoid having to change references within portfolio files. This first stage is therefore recommended as a way of simplifying the exchange between systems.

Typically a learner's information is stored in a database; the LIP-i is created as needed for transport. Each LIP-i and its associated files are to be prepared as follows:

1) Parse the IMS Learner Information instance document (or examine its source) to discover files that are locally referenced and need to be included in the Package. (see: `LIP activities.products…`)

2) Place the files that are to be included in the Package in the allocated sub-directory.

3) If any of the referenced files themselves make relative references to further files (e.g., an HTML document references some .gif files in a relative images/ directory), then these additional files also need to be included, preserving their relative directory structures, if any, within the allocated sub-directory. (This is to avoid having to change file references within the portfolio files themselves, which could be in binary rather than HTML format).

4) To the LIP-i being created, add (or change if necessary) references to the included files. These references should be relative to the sub-directory allocated to the LIP-i in the Package.

Note: Files that are externally referenced via a URL, and hence not included within the Package, are left UNCHANGED in the Learner Information File.

5) Add the Learner Information instance to the top level of the allocated sub-directory

A sub-directory allocated to a LIP-i might have the following form:

```
LIP.xml
MyPix.jpg
Transcript.pdf
Logo.jpg
Project1.html
Project1_images/tree.gif
Project1_images/bush.gif
Project1_images/shrub.gif
```

LIP.xml is the LIP-i at the sub-directory's root level. MyPix.jpg, Transcript.pdf, Logo.jpg, & Project1.html are root level files. "Project1_images" is a sub-directory with resources for, and referenced within, the Project1.html page. (HTML is used here as a general example). Portfolio files at the same level as the LIP.xml file are referenced within it as simple file names, relative to their shared root directory.

2.3.1 Files Common to Multiple Instances

When a common file is referenced by many instances in a Package (e.g., Logo.gif), there are several alternatives which may be used:

- Use the new <dependency> sub-element introduced in Content Practice v1.1 (See: CP Information Model 3.1.6, and CP BPG 4.5 <resources> Element). This allows a file or linked set of files, used by several resources, to be held under a single <resource> element and referenced by the client <resource> elements using a <dependency> sub-element.
- Use an external file, referenced by a fully qualified external URL in a <file> element in each resource that uses it. This URL must be network accessible from the destination(s) to which the Package is being sent. This alternative could be used to indicate that the external file be downloaded and a copy stored locally to the instances. The instances' references to it may need to be modified to accord with its new location.
- The LIP-i themselves make internal, fully qualified references to the shared external resource. If this reference is to remain unchanged after unpacking, then using a <file> element as described above can be omitted, although the reference must still be accessible from the destination of the LIP-is. This alternative could be used to indicate that the external file reference within the LIP-i should be left as is, pointing to the file at its original location.
- Otherwise, the common file has to be replicated in the sub-directory of each instance that uses it and referenced in the <file> element of a resource.

2.4 Create and Add the Manifest for the Package

The Manifest file needs to provide an entry point to each Learner Information Package instance and its associated files. This is done by allocating a resource element to each within the manifest's <resources> section. Each LIP-i and each associated file is then referenced through a resource's <file> element with the following XML structure:

```
<resource identifier="LIP1" xml:base="LIP1/" href="LIP.xml">
  <file href="LIP.xml"/>
  <file href="MyPix.jpg"/>
  <file href="Transcript.pdf"/>
  <file href="Logo.jpg"/>
  <file href="Project1.html"/>
  <file href="Project1_images/tree.gif"/>
  <file href="Project1_images/bush.gif"/>
  <file href="Project1_images/shrub.gif"/>
</resource>
```

2.4.1 Identifying the Instance's Root Directory and the Instance File within It

It is essential to be able to identify the root sub-directory allocated to a Learner Information instance and its files. This is done by setting value of the <resource> element's xml:base attribute to the name of the allocated sub-directory.

It is also essential to be able to identify the IMS instance file itself. When the <resource> has multiple <file> elements, the href attribute of the <resource> element is used to identify this 'entrypoint', in this example referencing the LIP instance file, LIP.xml.

2.4.2 Creating the Resources Section of the Manifest

The Manifest file <resources> section is created as follows:

1) Create a new <resource> element within the <resources> section of the Manifest file.

2) Assign a unique value to the <resource> element's identifier attribute. e.g. <resource identifier="LIP1" …>

Note: When a CP <resource> element is used, an identifier is mandatory even if, as in this case, it may not be referenced from an <organization> element.
If the Package is to be aggregated with other IMS Packages, their manifests will be merged into a top level manifest. To be valid, all identifiers in this resulting Manifest must be unique. Therefore, unless there is control over the identifier naming scheme across all the Packages being aggregated, then it is recommended that all identifiers be globally unique identifiers (see Content Packaging BPG 4.8.1 Identifiers).
IMS released a document about unique identifiers titled: IMS Persistent, Location-Independent Resource Identifier Implementation Handbook.

3) This is a key step. Set the value of the <resource> element’s xml:base attribute to the name of the sub-directory allocated to hold the LIP-i and its files in the Package, for example:
```
<resource identifier="LIP1" xml:base="LIP1/" …>
```

4) Set the <resource> element’s href attribute to reference the LIP-i as the entrypoint.
```
<resource identifier="LIP1" xml:base="LIP1/" href=“LIP.xml”>
```

5) Create a file reference for the LIP-i, for example:
```
 <file href=“LIP.xml”/>
```

6) Create a file reference for each of its associated files, for example:
```
 <file href=“MyPix.jpg”/>
```
The file reference is relative to the base address. If a file included in the sub-directory has a further sub-directory structure then this sub-path is included, such as:
```
 <file href=“Project1_images/tree.gif”/>
```

7) Repeat until all of the LIP-is and their file sets have been referenced by corresponding <resource> elements, creating one resource for each LIP-i.

Note: While both a LIP-i and a <resource> or a <file> element can reference a fully qualified URL that is external to the Package, a learner profile can be expected to have a long lifecycle. Files that are important to it are therefore better packaged and passed with it, unless there is a guarantee that they will persist at their referenced location.

2.4.3 Optional Use of Organizations

LIP-i <resource> elements may be optionally structured in the <organizations> section of the Manifest, by using a table of contents:

```
<organizations default="LIPS">
  <organization identifier="LIPS" title="default">
    <item identifier="new" title="New York Profiles">
      <item identifier="LIP1" identifierref="LIP1"/>
      <item identifier="LIP2" identifierref="LIP2"/>
      <item identifier="LIP3" identifierref="LIP3"/>
    </item>
    <item identifier=”update” title=”UpState Profiles”>
      . . .
    </item>
  </organization>
</organizations>
```

This use of the <organizations> section is for convenience, and is not required.

The <organization> section could be used, as in the example, on an application-specific basis, to group alumni according to state, learners by classes, and so forth. The meta-data section of the <organization> element or the <item> element (CP v1.1) may be used to provide appropriate information.

Other possible uses of the <organization> element might include grouping instances created using more than one IMS specification.

For example, a course creation tool might want to use a Content Package to pass course information to a course catalog system. However, it may be desired to also pass information about the originating department, the staff responsible, and other information which would best be carried in an instance created using IMS Enterprise. The two could be passed together by having two top level <item> elements grouped together by an <organization> element, or under a single <item> element. One <item> element would reference a <resource> element for the Enterprise instance, the other <item> element being a top level node for the course information expressed as a normal Content Package <item> hierarchy.

Further uses are suggested by the observation that a live class associates both a class membership and a course structure together with learning resources. These might include moving a live class to another system, communicating it to another system to be used for a specialized activity, or providing support for nomadic/mobile learning.

In general, the use of the <organization> element may be worth exploring as a flexible means of associating instances of different specifications for purposes that do not require the use or justify the creation of a specialized XML Schema or DTD.

2.5 Transfer the Package File Directory to the Transfer Container

The final step involves transferring the entire directory structure into some container for transportation. One way of doing this is to consolidate the entire directory structure, including all files, into a single file. The default format suggested in CP 1.1 is ZIP, but others such as CAB, JAR, and TAR can also be used. An alternative method, increasingly being adopted by other specifications (e.g. ebXML, BizTalk Framework 2.0, SOAP with Attachments), is to use MIME, in particular MIME/related, as a packaging format. This is more general as a mechanism for messages that transfer multiple files and does not exclude using a compression technique for contained files. It could also be used to send multiple Content Packages in a single message. However, the Content Packaging specification does not specify a packaging format, so this is left open here also.

Whatever transfer encoding/format is used, it is important that *the directory structure be maintained*, as the directory structure provides distinctions among different files with the same names.

The content packaging file is now ready for transport between systems.

Note: The content packaging specification, and this application of it, do not constitute a messaging system. Messaging is outside of the scope of the IMS Content Packaging specification.

3. Recommendations for Local Storage of Learner Portfolio Files

Reflecting on the inherent need for Learning Information Files to be moved between different organizations and sites, suggests that thought should be given to how portfolio and other related files are stored, organized, and maintained at a site. To simplify the task of transportation and exchange, the following suggestions are offered:

- Where possible, refer to common files via a URL that can be openly accessed over the Internet, e.g., an organization's logo file.
- All portfolio files should be coherently organized under a single root directory, or where not, be given URLs that can be openly accessed and then referenced within the LIP-i.
- All cross-references between files in a portfolio should be relative to the given root directory (similar to treating each user as their own web site).

Where these are adhered to, it should make the task of exchanging Learner Information Package collections much easier. Indeed, it should only require the IMS Learner File to be created and added to the learner's portfolio root sub-directory and the whole sub-directory is then ready to be packaged into the exchange format.

4. Unpackaging a Learner Information Content Package

The Learner Information Content Package is unpacked by using the Manifest file. If used, the organization section may define some categorization of the LIP-is. Ignore the organization if its use is not defined by the participants in the transaction. The process of unpackaging is as follows:

1) Open the IMS Package, and expand it to recreate the original directory and file structure.

2) Select a <resource> element. Typically this is done using the resources section and working down though the list of <resource> elements. Alternatively, if a Table of Contents is available in the <organization> section, this may be used to direct the selection process.

3) Retrieve each of the <resource>'s files by obtaining the base address from the <resource>'s xml:base attribute and prefixing it to the filename found in each <file> element's href attribute.

4) The files should then be stored locally ensuring that the relative file structure is maintained.

Note: If the recommendation in Section 3 of this implementation handbook is to be followed, then the entire directory can simply be copied to storage in one operation. However, the LIP-i and the <file> elements in the resource should be scanned for references to files outside its directory and checked for accessibility from the final location. They should also be checked for references to files not held in its root sub-directory, i.e. for cases where this handbook is not being followed.
A policy for unpacking and storing common files held in <dependency> elements, if used, also has to be established.

5) Populate the Profile database with information from the LIP-i. Make sure that portfolio file references in the new profile record take account of any changes made when transferring associated files to local storage.

6) Repeat until all LIP-is and their files have been retrieved and the contents stored.

7) Delete any temporary directory structure created for unpackaging purposes.

The root sub-directory and its associated base address, created for each instance when adding its files to the Package, is typically removed before transferring to storage, and a new local base address created.

If consistent practice by both parties is maintained for the local storage of portfolio files as suggested above, it should be possible to simply to unpack the portfolio directory and file structure in the Package directly under the local site's portfolios root directory, perhaps assigning a locally unique learner name to each portfolio's root sub-directory, with all relative addresses being maintained unchanged.

But, ultimately, decisions as to how this process is managed, how files are stored, and how the Learner Information record is linked to its associated files is implementation-dependent and are the responsibility of the implementer and the implementing site.

5. The Simplified Case without Associated Files

Packaging may be simplified when none of the LIP-is refer to any files contained within the Package. If there are no files other than the LIP-is, then uniquely naming the files is a simpler method for aggregating LIP-is than creating a uniquely named directory for each. The Content Package would then have the form of:

```
imsmanifest.xml
LIP1.xml
LIP2.xml
LIP3.xml
...
```

The steps are:

1) Add the IMS Learner Information instance documents to the Package's file area. No sub-directories are needed. The Package then includes the following files:

```
LIP1.xml
LIP2.xml
LIP3.xml
...
```

2) For each IMS Learner Information instance, create a new <resource> element within the Content Package Manifest's <resources> element

3) Create a reference to the IMS XML instance file using the <file> element of the <resource>.

```
<resources>
  <resource identifier="LI1">
    <file href="LIP1.xml"/>
  </resource >
  <resource identifier="LI2">
    <file href="LIP2.xml"/>
  </resource >
    ...
    ...
</resources>
```

The resources section of the manifest file has one <resource> element for each LIP-i file included in the Package.

Because no directory is used, and as the <resource> element has no base address, unpackaging systems should be able to uniformly interpret both simple and compound Learner Information Content Packages.

The <organizations> section may optionally be used, as described for the compound case.

The simplified case is unpackaged in the same manner as the compound case, above. The directory structure is completely flat and the only files that are retrieved are IMS LIP-is. The base is omitted, but that should not affect processing, as the default is the root directory.

6. Using this Approach to Package any Other IMS Instances

Given that this approach does not involve incorporating the data of other IMS XML instances into the Manifest itself, but treats them as separate files to be included within the Package, this method for using the IMS Content Packaging specification can equally serve to package instances of any other IMS specifications and their supporting files, if any. As with the Learner Information, the existing IMS Content Packaging specification does not need to be changed in any way to support other IMS specifications and therefore this extended usage likewise has no impact on the Content Package's existing use with content.

7. Using Resource Type to Indicate the Type of IMS Instance

When unpacking a Package of IMS instance files at the receiving end, it is valuable to have information about the type of file being passed without having to first open it and examine the header to determine this. Having a type label also makes it easier to mix instances of different IMS specifications in the same Package, should this be desired. It also makes it easier to mix QTI instances with other content in a Package. Treat a QTI instance as another file to be packaged. This approach should be used in preference to embedding QTI XML into the manifest itself, as was suggested in Content Packaging v1.0.

To ease the Package processing task, it is therefore recommended that the <resource> element's "type" attribute is used to indicate the type of IMS instance that it references. The following table sets out a recommended set of type labels to refer to the various IMS specifications, bindings, and version numbers. They take the general format of:

```
"ims"spec _ binding _ version number point release
```

IMS Specification	**Type label**
Accessibility v1.0	*imsacc_xmlv1p0*
Competency v1.0	imsrcd_xmlv1p0
CP v1.0	imscp_xmlv1p0
CP v1.1	imscp_xmlv1p1
CP v1.1.1	imscp_xmlv1p1p1
CP v1.1.2	imscp_xmlv1p1p2
Enterprise v1.1	imsent_xmlv1p1
Learning Design v1.0	imsld_xmlv1p0
LIP v1.0	imslip_xmlv1p0
Meta-Data v1.1	imsmd_xmlv1p1
Meta-Data v1.2	imsmd_xmlv1p2
Meta-Data v1.2	imsmd_rdfv1p2
QT1 v1.0	imsqti_xmlv1p0
QT1 v1.1	imsqti_xmlv1p1

Expressing version and point releases will use a consistent mechanism (shown above) of using "v" followed by the version number and "p" followed by the point release number. This should mean that the recommended IMS type labels can be easily maintained.

Although most IMS specifications have only provided an XML binding, this may not always be the case. For example, an RDF binding was produced for IMS Meta-Data v1.2, in addition to the normal XML binding. Others bindings for other specifications may be developed in the future.

It is likely that the Accessibility Working Group will only produce a Best Practice Guide and therefore not generate its own instances. However, the opportunity is taken to reserve these terms should they be needed in future.

In general, all Type labels beginning with "ims" are reserved.

Appendix A – Additional Resources

IMS Documents

IMS Content Packaging specification:
http://www.imsglobal.org/specificationdownload.html

IMS Persistent, Location-Independent Resource Identifier Implementation Handbook
http://www.imsglobal.org/implementationhandbook/

Appendix B – List of Contributors

The following individuals contributed to the development of this specification:

Adam Cooper	Fretwell-Downing
Bill Olivier	CETIS
Colin Smythe	IMS
Tom Wason	IMS

About This Document

Title	Using IMS Content Packaging to Package Instances of LIP and Other IMS Specifications Implementation Handbook
Editors	Bill Olivier, Mark McKell
Version	1.0
Version Date	August 2001
Status	**Final Handbook**
Summary	This document describes how to use IMS Content Packaging to assemble single or multiple instances of LIP, or other IMS specifications, together with any associated files, into a Content Package for transporting between systems.
Revision Information	9 August 2001
Document Location	http://www.imsglobal.org/implementationhandbook/

Revision History

Version No.	Release Date	Comments
Base 1.0	3 May 2001	Based on draft of document from 7 March 2001.
Final 1.0	9 August 2001	Made general wording and editing changes and made amendments for the use of: a) IMS Persistent, Location-Independent Resource Identifiers. b) The new <dependency> element for shared files. c) the <resource> “type” attribute to indicated the type of IMS file and a recommended vocabulary.

Index