The TMS Interoperability Protocol Package (TIPP)

V1.3.0 Draft 3

Document Version 3

Introduction 4

Why TIPP? 4

Relationship to Other Efforts 4

Goals and Non-goals of TIP and TIPP 5

Versioning of this Reference 5

Glossary of Terms 6

Basic Structure 7

Package Description File (manifest.xml) 7

Package Object Container (pobjects.zip/pobjects.zip.enc) 7

Package Object Sections 7

Ordering of Package Objects 9

Request and Response Packages 10

Transitivity 10

Security 10

Processing Expectations 10

TIP Lifecycle 10

Translate Request Packages 11

Review Request Packages 12

QA Request Packages 12

Quote Request Packages 12

Version specific Information and limitations 12

Version 1.3.0 12

Reference Guide 13

Naming convention for files 13

Naming Restrictions 13

Tool Identifiers 13

Communication Endpoint Identifiers 14

Format of Date/Time Fields 14

Format of Package Object Paths 15

See also 16

Standards 16

TEMP: Misc. Questions 16

TEMP: Decisions and Argumentations 16

# Document Version

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **TIPP Ver.** | **Date** | **Author** | **Changes/Decisions** |
| 2 | 1.0.1 | 14/11/10 | Sca | D: non transitive package D: Versioning convention D: IDs |
| 3 | 1.2 | 26/6/11 | Chase Tingley | Align with schema updates |
| 4 | 1.2 | 28/6/11 | Chase Tingley | Updated naming (TIPP), added Relationship to Other Efforts section |
| 5 | 1.3.0 | 19/8/11 | Chase Tingley | Updated based on feedback from Yves Savourel, Christian Lieske, and David Filip. |

# Introduction

The TMS Interoperability Protocol Package (TIPP) is an information container that allows the seamless exchange of information between different independent Translation Management Systems (TMS).

This reference guide describes the package, the package description and standard methods to interact with the package. The representation of the content itself is not described in this document.

## Why TIPP?

There exist two major ways of exchanging content between TMS:

* Proprietary solutions between TMS of the same vendor that support content as well as additional information.
* Different file formats standards that usually only representation guides for content and not for metadata and process information.

In more and more use-cases, different TMS are used within the same process. This situation applies because either the translation buyer uses another system than the translation vendor, or within large buyer and/or vendor infrastructures different TMS are employed for various reasons.

The TIPP enables different TMS to exchange and to deploy the various package objects contained in the container in an automated way. Besides localizable files the container may also hold additional information and data files:

* Information:
  + Process related information
  + Content descriptions
  + Tool related information
  + Metrics
* Files:
  + Translation Memory (TM)
  + Terminology
  + Reference material
  + Style guide

## Relationship to Other Efforts

Exchanging and interchanging various types of data between different translation management systems (TMS) is an area of increasing activity in the field of product and media localization and translation. It comprises multiple workflows of human and machine processes, multiple data types, and multiple technologies. The Translation Interoperability Protocol (TIP) is designed to enable the seamless and lossless sharing of data and information between different independent TMS based on open standards for data representations. The technical means for this sharing is the TIP Package (TIPP), which represents a transport container with several resources (TIPP Objects) that are needed to accomplish a certain translation task.

Currently, there are several other efforts under development related to container formats for translation interoperability. We consider those efforts to be complementary to our own approach. The focus of the initial version of TIPP is to document and implement a package format that is machine-readable and can reliably exchange translatable resources and certain limited types of metadata.

In contrast, other approaches are tackling efforts to describe additional business related processes and metadata, for example, from bidding through actual translation tasks to the final evaluation and billing procedures, at the expense of complexity and machine readability. We believe that both efforts may benefit from additional collaboration and cross-pollination of ideas in the future.

## Goals and Non-goals of TIP and TIPP

The purpose of TIP is to provide a way to exchange data that has been normalized for translation between different links in a translation tool chain. Because of this focus, there are a number of broader problems that it does not attempt to solve. In particular, TIP does not attempt to define the boundary between CMS systems and translation tools; it assumes that its translatable content has already been imported into a TIP-aware system somehow.

More generally, it does not attempt to address use cases involving the needs of systems outside of the translation tool chain. For example, some localization workflows expect files or directories to be renamed based on locale-dependent (or other variable) elements. These use cases are beyond the current scope of TIP.

Even within these constraints, the present scope of TIPP is narrow. This version supports only bilingual translation projects, and does not support the partial completion of a project. These use cases may be supported in future iterations of TIPP.

## Versioning of this Reference

An important part of the approach for the TMS Interoperability Protocol Package is an agile und fast implementation of the first version of TIPP. This reference might therefore change quickly over time and it is important to understand the compatibility between the versions.

The version of this reference (which will also be included in every package) consists of three numbers, period-separated, starting with 1.0.1.

The first number describes the major version: Differences between major versions might lead to a lack of compatibility.

The second number describes the version of the Package Description File Format: All basic changes in the manifest.xml will lead to a new second digit.

Changes in the third number apply for the changed support of content types within the package without changing the basic structure of the manifest.xml.

# of Terms

|  |  |
| --- | --- |
| Envelope | The outermost container in a TIPP package. In the current implementation, this is a ZIP archive. |
| Package Description File | An XML file that contains all metadata needed to process a TIPP package. |
| Package Object Container | The portion of a TIPP package that stores the package objects. The Package Object Container is represented as an (optionally encrypted) ZIP archive located within the Envelope. |
| Package Object | An individual file stored in a TIPP package. |
| Request | A TIPP package that defines a particular localization task to be completed. |
| Response | A TIPP package that contain the results of an attempt to complete the localization task defined by a given Request. |

# Basic Structure

A TIP Package consists of two parts:

* The Package Description File, an XML file defined by the TIPP Schema.
* An (optionally encrypted) Package Object Container. The Package Object Container is a ZIP archive containing one or more Package Objects.

These two parts are contained in the package itself, which is also called the Envelope. In the current implementation, the Envelope is a ZIP archive.

Description File

Object Container

(optionally encrypted)

There is no folder structure within the Envelope. The Package Description File must be named manifest.xml. The Package Object Container which must be named pobjects.zip.enc or pobjects.zip depending on if it’s encrypted or not.

## 

## Package Object Container (pobjects.zip/pobjects.zip.enc)

### Package Object Sections

The Package Object Container contains a folder for each of the different content types it can include. These folders are called Package Object Sections. Each Section is identified by its folder name.

**Note:** In version 1.3.0, only the *bilingual*, *input*, *output*, and *tm* sections are allowed by the schema. Information about all other sections should be considered provisional at this point.

| Folder | Description | File Types | Supported in v1.3.0? |
| --- | --- | --- | --- |
| bilingual | Contains content to be processed in bilingual file formats. As bilingual files encapsulate their own translations, these files are edited in place. There may be multiple files in this folder, each referenced in the *manifest.xml* file. | XLIFF:Doc | Yes |
| Input | Contains source content to be processed. Files in this folder are considered read-only; as the files are processed, they are written back to the *output* folder. There might be multiple files in this folder, each referenced in the manifest.xml file. | Any localizable | Yes |
| output | Contains target content that has been processed content. Each file in this folder should correspond to a single file in the *input* folder. There might be multiple files in this folder, each referenced in the manifest.xml file. | Any localizable | Yes |
| Tm | Contains Translation Memory reference material. There might be multiple files in this folder, each referenced in the manifest.xml file. | TMX, XLIFF | Yes |
| Term | Contains Terminology reference material. There might be multiple files in this folder, each referenced in the manifest.xml file. | TBX, OLIF | No |
| reference | Contains general reference material. There might be multiple files in this folder, each referenced in the manifest.xml file. | Any | No |
| Workflow | Contains workflow related information. There might be multiple files in this folder, each referenced in the manifest.xml file. | Any | No |
| Metrics | Contains metrics related information (word counts, etc.). There might be multiple files in this folder, each referenced in the manifest.xml file. | Any | No |
| attachment | Contains additional attachments related to the localization process, such as style guides, contact lists, or other types of information. Files in the *attachment* folder are not assumed to be machine readable. There may be multiple files in this folder, each referenced in the *manifest.xml* file. | Any | No |

All Sections are optional, except as described under . If a content type is not used, the folder may be omitted.

Package Object Sections can support subdirectories.

### Ordering of Package Objects

The objects within a section may be assigned a positive integer as a sequence number. Sequence numbers provide information about the intended order of processing for the objects in that section. For example, if there are multiple bilingual files in the *bilingual* section, they should be presented to the user for translation in the order specified by their sequence numbers.

## Request and Response Packages

Packages are classified as either Requests or Responses. The Package Definition File for a Request TIPP describe a localization task to be performed, along with all necessary TIP information to perform the task. A Response package contains information about the results of an attempt to process a given Request.

A Response package is a distinct package with its own unique identifier. It embeds the ID of the Request package to which it responds as part of its Response metadata.

A Response package reports either "Success" or "Failure" of the task described in the referenced Request package.

## Transitivity

The first version of the TIPP will not support transitive packages. Each package will be generated exclusively for one unidirectional communication path and has reached the end of its lifecycle after it has been unpacked and processed. For the return path or a forwarding path, new packages will be generated. A reference ID is used to identify returning packages.

In a future version, it is planned to support transitive packages, that can flow through complex communication paths.

## Security

To ensure security, the Content Package can be encrypted.

The encryption is done based on Public-key Cryptography and uses the RSA algorithm. .

# Processing Expectations

## TIP Lifecycle

Between Request and Response, any processing on the package level is opaque to the originating system. The package may be processed by a single other system, or possibly more than one - there may even be additional TIPP sub-lifecycles between the processing nodes. In this release, all of this is invisible to the generating package. There is no concept of package transitivity and no support for recording information about what systems have processed the package contents, other than the one system that generates the Response package.

However, the originating system has certain expectations about the contents of a successful Response package. These expectations vary by Request type.

### Translate Request Packages

A system that generates a Translate Request package expects a basic localization process to have been performed on the package contents that it receives back as a successful Translate Response. The expectation for a particular package object depends on the location of the object in the package.

There are no expectations regarding package contents for Translate Response packages that report failure. The ResponseComment element may be optionally populated by the responding system in order to provide more information about the failure.

#### Bilingual Objectss

By default, objects in the *bilingual* folder are expected to be processed for localization in place during a successful Translate action. The precise meaning of this processing is beyond the scope of this specification and should be defined on a per-format basis. The localizable package attribute may optionally be used to disable this expectation. If a *bilingual* objects has the localizable value of "no", it is expected that no localization will be performed on this file.

#### Input Objects

The *input* folder contains all non-bilingual, localizable objects in the packages. Once created as part of a Request package, it should be considered read-only. However, Response packages may omit the contents of the *input* folder to save space, provided they do not alter *input* section of the Package Description File itself.

#### Output Objects

In a Translate Request package, the output folder should be empty and may be omitted.

In a Translate Response package, the contents of *output* represent the output of the translation process for non-bilingual files. For each object listed in the *input* section of the Package Description File, a Response package should contain a corresponding file in its *output* folder.

When the originating system receives a successful Response package, it expects that for each object in the *input* folder, there will now be a corresponding object in the output folder to represent. Each localized object in *output* should have the same LocationPath value as the *input* object to which it corresponds.

The objects in *output* represent the localized form of the *input* objects. What this means varies both by file type, as well as by the value of the optional localizable attribute. *input* objects with a localizable value of "no" should be copied intact to the *output* folder; no additional processing is required. *input* resources with a localizable value of "yes" should be localized according to file type.

Successful Response packages that do not contain a corresponding *output* object for each *input* object should be considered invalid. In other words, there is no support for partial completion of Translation tasks.

### Review Request Packages

This task type is not supported in this release.

### QA Request Packages

This task type is not supported in this release.

### Quote Request Packages

This task type is not supported in this release.

# Version specific Information and limitations

## Version 1.3.0

This version has the following limitations:

* Encrypted Package Object Containers are not supported. All package Envelopes are expected to contain only an un-encrypted *pobjects.zip* in addition to the Package Description File.
* The only Package Object Sections that are supported are *bilingual, i*nput, output and *tm*.
* The only type of Package Object that can be included in a TIPP is a file that is directly embedded in the Package Object Container. Future releases may allow references to external releases (for example, reference material accessed via HTTP).
* No tool-specific features are supported.
* Unique IDs are created by the tools and not through a centralized method.
* The only supported Request type is *Translate*.

# Reference Guide

## Naming convention for files

Envelopes should be identified by the suffix **.tipp**. The contents of the Envelope should be named as follows:

|  |  |
| --- | --- |
| **Name** | **Description** |
| manifest.xml | Package Description File |
| pobjects.zip | Package Object Container (un-encrypted) |
| pobjects.zip.enc | Package Object Container (encrypted) |

### Naming Restrictions

In order to minimize platform-specific incompatibilities, both Envelopes names and all Package Objects paths are restricted to the following subset of ASCII:

* a-z
* A-Z
* 0-9
* Underscore ('\_'), dash ('-'), period ('.'), or space (' ')

Each object name may contain a maximum of 240 characters.

## Tool Identifiers

Information about what tools generate the task and response packages are encoded in the package manifests. In the time prior to the availability of centralized repositories and functionalities tied to tool identity, this information is considered informational. There is currently no mechanism for a task package to require that a particular tool be used to process it and generate the response.

Tools are described by the ContributorTool element, and encode three pieces of data:

* The common name for the tool
* The tool ID, expressed as a URI
* The tool version, expressed as a string

For example, the common name for a tool might be “GlobalSight”, with version “8.1” and ID “<http://www.globalsight.com>". For now, the specific semantics of Tool IDs are left up to the tool makers.

## Communication Endpoint Identifiers

Information about the systems that generate the task and response packages are encoded in the package manifests. There is currently no mechanism for a task package to require that a particular endpoint be used to process it and generate the response.

Communications endpoints are identified by three pieces of information:

* The common name for the endpoint, such as the name of the controlling organization
* The endpoint ID, expressed as a URI, such as the URI of the specific system that generated the package
* A timestamp, recording the time when the package was created according to [Format of Date/Time Fields](#_Format_of_Date/Time).

The elements used to describe the endpoint differ for task and response packages. Task packages describe the originating endpoint in the PackageCreator section, using the CreatorName, CreatorID, and CreatorUpdate fields. Response packages describe the responding endpoint in the OrderResponse element, using the ResponseName, ResponseID, and ResponseUpdate fields.

|  |  |  |
| --- | --- | --- |
| **Field** | **Originating endpoint** | **Responding endpoint** |
| Name | CreatorName | ResponseName |
| ID | CreatorID | ResponseID |
| Timestamp | CreatorUpdate | ResponseUpdate |

Additionally, Communication Endpoint sections include the Tool Identifier of the tool that process the TIPP at that endpoint.

## Format of Date/Time Fields

Several *manifest.xml* fields contain date and time data. All of these fields use a fixed format to encode their data. The format consists of year/month/day information, followed by the string literal ‘T’, followed b hour/minute/second information:

YYYY-MM-DDThh:mm:ss

All times are UTC. Other date/time formats are treated as errors.

## Format of Package Object Paths

All ObjectFile elements specify the location of an object in the package via their LocationPath child element. The value of LocationPath must follow the following rules:

* The path is relative to the top-level package folder corresponding to the containing PackageObjectSection. The path should not include the name of this folder. For example, for ObjectFile elements with an “input” PackageObjectSection, all LocationPath values are considered relative to the “input” folder in the package, and do not need to be prefixed with “input”. Similarly, path values should not be prefixed with “/”.
* All paths are considered case-sensitive.

If the package creator creates additional folder structure beneath the top-level package folders, additional rules exist to govern references to objects within these subfolders:

* The forward slash (“/”) is used to separate path components.
* All paths must be normalized into a canonical form consisting solely of named path elements and path separators. The path elements “.” and “..” are not supported.

# See also

## Standards

|  |  |  |
| --- | --- | --- |
| Name | Organization | Description |
|  |  |  |
|  |  |  |
|  |  |  |



# TEMP: Misc. Questions

This is for questions we want to track during the development of the reference guide, which are not captured elsewhere in the document.

* Does "TIPP Version" encapsulate "TIP Version"? In other words, does the knowledge that a TIPP package is version 1.3.0 make any guarantee about the version of any xliff:doc objects it might contain?
  + If independent versioning is allowed, should TIPP manifests include version information about the xliff:doc files on a file-by-file basis?
* For this version, should we allow XLIFF as a TM carrier format?
* Will we ever support bilingual file formats in addition to xliff:doc? (PO, regular XLIFF, etc)
* What formats will be officially supported in the package sections that aren't allowed yet? (eg: terminology, metrics, etc)

# TEMP: Decisions and Argumentations

This is to keep track of the ‘why’ in some decisions made as part of the process.