The TMS Interoperability Protocol Package (TIPP)

V1.2 Draft 2

Document Version 3

Introduction 4

Why Use TIPP? 4

Relationship to Other Efforts 4

Versioning of this Reference 5

Basic Structure 5

Packing Mechanism 5

TIPP Description 6

Package Object Sections 6

Package Object Sequences 8

Package Description File (manifest.xml) 8

Transitivity 8

Security 8

Version specific Information and limitations 9

Version 1.2 9

Processing Expectations 9

Package Lifecycle 9

TaskType: Translate 9

TaskType: Review 11

TaskType: QA 11

TaskType: Quote 11

Reference Guide 11

Naming convention for files 11

Tool Identifiers 11

Communications Endpoint Identifiers 11

Format of Date/Time Fields 12

Format of Package Object Paths 12

See also 14

Standards 14

Glossary 14

TEMP: Misc. Questions 14

TEMP: Decisions and Argumentations 15

# 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 |

# 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 Use 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 the actual content that needs to be translated 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-processable and can reliably exchange translation assets 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.

## 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 digits starting with 1.0.1.

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

The second digit 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 digit apply for the changed support of content types within the package without changing the basic structure of the manifest.xml.

# Basic Structure

## Packing Mechanism

The TIPP contains optionally encrypted information as well as unsecured meta-information. To ease the handling of the content after decryption, the encryption is applied to a complete archive within the package.

The basic structure of the package is

TMS Interoperability Protocol Package

Object Container

(optionally encrypted)

Description File

### TIPP Description

The Enclosing Package only contains the Package Description File and the Package Objects in an optionally encrypted container.

There is no folder structure within the enclosing package; the only content is the Description File which must be named manifest.xml and the Package Objects Container which must be named pobjects.sec or pobjects.zip depending on if it’s encrypted or not.

### Package Object Sections

The 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.

| Folder | Description | File Types |
| --- | --- | --- |
| 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 |
| 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 |
| 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 |
| tm | Contains Translation Memory reference material. There might be multiple files in this folder, each referenced in the manifest.xml file. | TMX, XLIFF |
| term | Contains Terminology reference material. There might be multiple files in this folder, each referenced in the manifest.xml file. | TBX, OLIF |
| reference | Contains general reference material. There might be multiple files in this folder, each referenced in the manifest.xml file. | any |
| workflow | Contains workflow related information. There might be multiple files in this folder, each referenced in the manifest.xml file. | any |
| metrics | Contains metrics related information (word counts, etc.). There might be multiple files in this folder, each referenced in the manifest.xml file. | any |
| 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 |

All content is optional, except as described under . If a content type is not used, no folder should be created.

### Package Object Sequences

The files in a given Section may be divided among multiple logical groups, called Package Object Sequences. A Section may divide its contents into multiple Sequences to represent concepts such as distinct workflow tasks or other sub-package groupings. Each Sequence is identified by a positive integer that must be unique within its containing Section. (Sequence ids do not need to be unique across the entire package.)

By default, all non-empty Sections consist of a single Sequence that contain all the objects in that Section.

## Package Description File (manifest.xml)

The Package Description File contains information on all Package Objects included in the Container. Any Objects not described by the Container can be ignored by package processors.

## 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. .

# Version specific Information and limitations

## Version 1.2

This version has the following limitations:

* Encrypted package contents are not supported. All packages are expected to use *pobjects.zip*.
* Only *bilingual, i*nput, output and *tm* are supported as PackageObjectSection types.
* The only type of PackageObjectsupported is ObjectFile.
* No tool-specific features are supported.
* Unique IDs are created by the tools and not through a centralized method.
* The only supported TaskType is *Translate*.

# Processing Expectations

## Package Lifecycle

The TIPP lifecycle is initiated by an originating system, which creates a TIPP package containing some unit of work and a OrderTask of one of the supported types. This package is referred to as a Task package. When this work is completed, another package is returned to the originating system, containing metadata that identifies an existing Task package, along with a valid OrderResponse. This package is referred to as a Response package.

Whatever processing takes place in between Task and Response is opaque to the generating 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 task type.

### TaskType: Translate

When an originating system issues a package with TaskType "Translate", it expects a basic localization process to have been performed on the package contents when it receives a Response with the ResponseMessage value of "Success". The expectation for a particular resource depends on the location of the resource in the package.

There are no expectations regarding package contents for Response packages with the ResponseMessage value of "Failure". The ResponseComment element may be optionally populated by the responding system in order to provide more information about the failure.

#### Bilingual Resources

By default, resources 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. For XLIFF:doc files in the bilingual folder, it is expected that all trans-units will contain valid targets. The localizable package attribute may optionally be used to disable this expectation. If a *bilingual* resource has the localizable value of "no", it is expected that no localization will be performed on this file.

#### Input and Output Resources

The *input* folder contains all non-bilingual, localizable resources in the packages. Once generated by the originating system, it should be considered read-only. Since the *output* folder in a Translate package should be empty, it should be omitted from the package. When the originating system receives a successful Response package, it expects that for each resource in the *input* folder, there will now be a corresponding resource in the output folder to represent. Each localized resource in *output* should have the same LocationPath value as the *input* resource to which it corresponds.

The resources in *output* represent the localized form of the *input* resources. What this means varies both by file type, as well as by the value of the optional localizable attribute. *input* resources 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.

Response packages must contain all of the *input* resources that were contained in the original Task pacakge. Successful Response packages that are missing *input* resources from the original Task package should be considered invalid. Additionally, Successful Response packages that do not contain a corresponding *output* resource for each *input* resource should be considered invalid. In other words, there is no support for partial completion of Translation tasks.

### TaskType: Review

This task type is not supported in this release.

### TaskType: QA

This task type is not supported in this release.

### TaskType: Quote

This task type is not supported in this release.

# Reference Guide

## Naming convention for files

Packages should be identified by the suffix **.tipp**. The tools can choose the other components of the package name.

## 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.

## Communications 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, and formatted 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 |

## 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.
* All paths must identify unique resources.

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 |
| XKMS | W3C | XML Key Management Specification (XKMS) uses the web services framework to make it easier for developers to secure inter-application communication using public key infrastructure (PKI). XML Key Management Specification is a protocol developed by W3C, which describes the distribution and registration of public keys. Services can access an XKMS compliant server in order to receive updated key information for encryption and authentication. |
|  |  |  |
|  |  |  |

# Glossary

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

# 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.

# TEMP: Decisions and Argumentations

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