Evolution of the PortageII API from PortageII 1.0 to PortageII 4.0

Task	PortageII 3.0/4.0 API	PortageII 2.2 API	PortageII 2.1 API	PortageII 2.0 API	PortageII 1.0 API
Figure out what version of PortageLive is on your server	3.0: getVersion() exists and returns "PortageII-3.0"; WSDL says PortageII 3.0 (2016) 4.0: getVersion() exists and returns "PortageII-4.0"; WSDL says PortageII 4.0 (2018)	WSDL Copyright is 2014 and method translatePlainTextCE() exists	WSDL Copyright is 2014, getTranslation2() takes four arguments and translatePlainTextCE() does not exist	WSDL Copyright is 2013 and getTranslation2() takes two arguments	WSDL Copyright is 2012 and translateSDLXLIFFCE() does not exist
List contexts on a PortageLive server	3.0: getAllContexts(verbose) 4.0: getAllContexts(verbose, json)	getAllContexts(verbose)	same	same	same
Preload models for a given system so it can run faster	primeModels(context, PrimeMode)	same	same	same	same
Translate one sentence or paragraph	translate(srcString, context, newline, xtags, useCE) In 3.0, translate() replaces getTranslation() and getTranslation2()	getTranslation2(srcString, context, newline, xtags)		getTranslation2(srcString, context)	
Translate a small collection of sentences or paragraphs (up to 10 or 20 sentences in total; the request must take less than 30 seconds in total or http timeouts will kill it)	translate() has a fixed overhead of .5 to 1 second per call. As a result, it is best to group queries if you can: construct a string with a newline between sentences or paragraphs; call translate() with it and set newline="s" or "p", respectively; the output can be split on newlines and matched 1-to-1 with input lines.	same as 3.0, calling getTranslation2() instead of translate(). The explicit control of the semantics of newlines in genTranslation() and getTranslation2() was introduced in 2.1 to facilitate the optimized translation of mini batches of sentences.		The output of grouped queries is not well aligned to the input, so loop over getTranslation2() one sentence or paragraph at time.	
Translate a large collection of sentences or paragraphs synchronously	Loop over the above procedure, 10 sentences or paragraphs at a time. Avoid submitting sentences one at a time if you can, to amortize the fixed overhead over larger blocks instead.	same as 3.0, calling getTranslation2() instead of translate().		loop over getTranslation2() one sentence at at time (expensive overhead: upgrade to a more recent version if speed is an issue).	
Translate a large collection of sentences or paragraphs asynchronously (preferred method; more efficient)	Construct a plain text file and call translatePlainText()	Construct a plain text file and call translatePlainTextCE()	n/a	n/a	n/a

Evolution of the PortageII API from PortageII 1.0 to PortageII 4.0 (p. 2)

Task	PortageII 3.0/4.0 API	PortageII 2.2 API	PortageII 2.1 API	PortageII 2.0 API	PortageII 1.0 API
Translate a plain text file asynchronously	translatePlainText()	translatePlainTextCE()	n/a	n/a	n/a
Translate a TMX file asynchronously	translateTMX()	translateTMXCE()	translateTMXCE()	translateTMXCE()	translateTMXCE()
Translate an sdlxliff file asynchronously	translateSDLXLIFF()	translateSDLXLIFFCE()	translateSDLXLIFFCE()	translateSDLXLIFFCE()	n/a
Get the status on an asynchronous translation request	translateFileStatus()	translateTMXCE_Status() or translateSDLXLIFFCE_ Status() or translatePlain TextCE_Status()	translateTMXCE_Status() or translateSDLXLIFFCE_ Status()	translateTMXCE_Status() or translateSDLXLIFFCE_ Status()	translateTMXCE_Status()
Transfer tags from source to target	All translate*() methods accept the xtags parameter. Set it to true to transfer tags found in the source text into the target language output.	Set xtags parameter to getTranslation*() or translate*CE()	Set xtags parameter to getTranslation*() or translate*CE()	Only supported for TMX and sdlxliff files	n/a
Enable confidence estimation (slow)	Models must be trained with CE; set useCE parameter to any translate*() method to get confidence estimates with your translations. In TMX and sdlxliff files, the CE score is in a proper field. With translate() or translatePlainText(), it is at the beginning of each line, but the output is only usable if you use nl="s".	All *CE() methods compute confidence estimates for models trained with CE. getTranslationCE() and translatePlainTextCE(), output only usable with nl="s".	same as 2.2, but not available for plain text files.	same as 2.1	same as 2.1
Upload / get / erase the fixed terms database for a model	updateFixedTerms() / getFixedTerms() / removeFixedTerms()	same as 3.0	n/a	n/a	n/a
Add a sentence pair to an incremental document model	3.0: n/a 4.0: incrAddSentence()	n/a	n/a	n/a	n/a
Get the status on incremental document model training	3.0: n/a 4.0: incrStatus()	n/a	n/a	n/a	n/a

Evolution of the PortageII plugin architecture from PortageII 1.0 to PortageII 4.0

PortageLive pipeline	PortageII 3.0 and 4.0	PortageII 2.2	PortageII 2.1, 2.0, 1.0
General note about the plugin architecture	As a result of significant improvements in the default PortageLive plugins with PortageII 3.0, you should review all customized plugins you use in your systems and determine if you can use the new default ones instead. The plugin architecture continues to allow ad hoc scripts, but most common problems we're aware of are now handled by default.	The plugin architecture is designed to accept ad hoc scripts to address specific problems observed; some standardized scripts address common problems.	The plugin architecture is designed to accept ad hoc scripts to address specific problems observed.
Clean up utf8 input	Script clean-utf8-text.pl now standardizes how to clean utf-8 input, both for training corpora (tmx-prepro) and in PortageLive (preprocess_plugin).	Clean up code exists in preprocess_plugin and tmx-prepro, but not fully consistent.	same as 2.2
Separate words joined with slashes (e.g., "inside/outside" -> "inside / outside") so they get translated properly.	Script fix-slashes.pl now used by default in preprocess_plugin.	Script fix-slashes available in preprocess_plugin; not enabled by default.	n/a
Handle numbers between French and English	By default, predecode_plugin uses mark-numbers-en2fr.pl or mark-numbers-fr2en.pl to perform precise rule-based translation of numbers between French and English.	Optionally (but not by default), predecode_plugin uses mark-english-number.pl to perform rule-based translation of numbers from English into French only.	fix-en-fr-numbers.pl can be used in postprocess_plugin to reformat numbers copied from English into French.

Evolution of the PortageII plugin architecture from PortageII 1.0 to PortageII 4.0 (p. 2)

PortageLive pipeline	PortageII 3.0 and 4.0	PortageII 2.2	PortageII 2.1, 2.0, 1.0
Insert non-breaking spaces in French output	By default, postprocess_plugin now uses add-fr-nbsp.pl to insert non-breaking spaces where required by French conventions: inside quotes, before currency symbols, etc.	postprocess_plugin has sample code for inserting some non-breaking spaces; not enabled by default.	same as 2.2
Insert non-breaking hyphens	postprocess_plugin has sample code for making some hyphens non-breaking; not enabled by default because encoding is not normalized and depends on the final document file format.	same as 3.0	same as 3.0
Translate hashtags in Tweets	We now have a specialized pipeline for handling hashtags and tweets from Arabic into English. Ask us if you'd like to enable this for your system and language pair.	n/a	n/a
Translate fixed terms using hard rules	Use the *FixedTerms API methods to set the rules, which are applied by predecode_plugin; Warning: this is not for your whole terminology database, only for absolutely unambiguous terms. The fixed-terms functionality was instroducted with PortageII 2.2, but its use is not generally recommended: use only to address specific and problematic errors your trained system makes.	same as 3.0	n/a

Evolution of the PortageII framework features and recommendations from PortageII 1.0 to PortageII 4.0

Task	PortageII 3.0 and 4.0 framework	PortageII 2.2, 2.1 and 2.0 framework	PortageII 1.0 framework
Need to retrain when upgrading from a previous version?	3.0: Yes. 3.0 includes important new features and many updates to the recommended training parameters: retrain all models from previous versions. 4.0: Recommended. Retraining is required to train or fine tune your own NNJM, and to enable incremental document adaptation. However, 3.0 and 4.0 can still run older models.	No. PortageII 1.0, 2.0, 2.1 and 2.2 models are all compatible. Exceptions: - a manual step is required to enable fixed terms for models trained before 2.2. - the soap-translate.sh file in PortageLive changed between 2.0 and 2.1.	Yes. This was a major update to the core MT engine, old models should be retrained.
Sparse features (DHDM)	The new sparse features, including the discriminitive reordering model, significantly improve translation quality and should always be used.	n/a	n/a
Coarse LMs	3.0: Enable Coarse LMs with 200 and 800 classes (this new features is enabled by default) 4.0: Use only if you don't fine-tune the generic NNJM or create your own NNJM.	n/a	n/a
BiLM	This new feature can sometimes improve system quality but is often not worth it; not enabled by default.	n/a	n/a
Use of Generic LM (MIXLM_ PRETRAINED_TGT_LMS)	We now recommend systematically using the Generic LM for all systems between French and English. It cannot introduce out-of-domain terminology, it can only help use your in-domain data better.	optional but recommended	
Use of Generic TM (MIXTM_PRETRAINED_TMS)	Use this model with your in-domain data to significantly increase the system's vocabulary up front, and potentially improve the quality of all your En<->Fr models.	same as 3.0/4.0	same as 3.0/4.0
HLDM	This baseline reordering model is always required, and should always be used, preferably in combination with the new, sparse one.	Introduced in 1.0, this powerful reordering model is always required.	

Evolution of the PortageII framework features and recommendations from PortageII 1.0 to PortageII 4.0 (p. 2)

Task	PortageII 3.0 and 4.0 framework	PortageII 2.2, 2.1 and 2.0 framework	PortageII 1.0 framework
NNJM	3.0: pre-trained NNJMs available with the generic models for fr->en and en->fr. 4.0: with small to medium corpora, we recommend you fine-tune the generic NNJM; with large corpora (2M sentence pairs or more), we recommend you train your own for best hybrid neural-statistical MT results. Requires a GPU; if you don't have one, use the pre-trained NNJMs, or Coarse LMs.	n/a	n/a
Significance Pruning	This process reduces the size of your phrase tables. Only use it for very large corpora, with 10M sentence pairs or more.	same as 3.0/4.0	n/a
Source-aware truecasing	Use source-aware truecasing whenever possible, i.e., as long as both your source and target language have the concept of case. This feature is automatically disabled for Chinese and Arabic.	same as 3.0/4.0	same as 3.0/4.0
Tuning with Batch Lattice MIRA	Introduced in 1.0, this state-of-the-art tuning algorithm is always recommended and is the default.		
Using IBM4 alignments	We now recommend systematically using IBM4 alignments, along with the usual IBM2 and HMM3 ones. This produces the most robust and reliable phrase tables.	optional	