１１．多言語処理

１１－１．概要

多言語処理は、同一の実行形式において、言語に依存するメニュー、ボタン等のダイアログ表示、エラー・メッセージ、ヘルプを、全て外部のテキストファイルから動的にロードすることにより、言語の切り替えを行う。

このことにより、デバッグや機能改善が行われた場合においても、再度プログラマによるビルド作業を行うことなく、その結果を各国語で利用できるようになる。

また、新たな言語に翻訳して利用しようとする場合においても、プログラム作業を行うことなく、翻訳家によりテキストファイルを用意するだけで対応が可能となった。

シミュレーションの機能を、安定性信頼性を追求した基幹部分と、機能拡張のための外部関数及びプラグイン DLL に分離し、新たなニーズへの対応を基幹部分と分離するとともに、多言語機能により言語毎にソースコードのバージョンが枝分かれした問題を解決するために、基幹部分を共通のソースコードに再統一し、完成度を高めることが可能となった。

多言語処理機能には、sim.exe を構成する各ダイアログのみならず、そこから起動される外部関数やプラグイン DLL をも協調的に動作させる仕組みも提供しており、更に外部関数やプラグイン DLL のコーディングを助けるための共用ライブラリも用意した。

システムが現在使用している言語は、表１１－１に示した ISO 639 のコード表により識別している。これを用いて、sim.exe 内部の各種ダイアログにおける表示言語の選択、セットアップにおける各種ファイルの配置、外部関数やプラグイン DLL における表示言語の協調などの制御を行っている。

表１１－１：ISO 639 の代表的な言語コード

言語	言語コード
アブハジア語	ab
アファル語	aa
アフリカーンス語	af
アルバニア語	sq
アムハラ語	am
アラビア語	ar
アルメニア語	hy
アッサム語	as
アイマラ語	ay
アゼルバイジェン語	az
バシキール語	ba
バスク語	eu
ベンガル語	bn
ブータン語	dz
ビハール語	bh
ビスラマ語	bi
ブルターニュ語	br
ブルガリア語	bg
ビルマ語	my
白ロシア語（ベラルーシ語）	be
カンボジア語	km
カタラン語	ca
中国語（簡体）	zh
中国語（繁体）	zh
コルシカ語	co
クロアチア語	hr
チェコ語	cs
デンマーク語	da
オランダ語	nl
英語	en
エスペラント語	eo
エストニア語	et
フェロー語	fo
ペルシャ語	fa
フィジー語	fj
フィンランド語	fi
フランス語	fr
フリジア語	fy
ガリシア語	gl
ゲーリック語（スコットランド語）	gd
ゲーリック語（マン島語）	gv
ジョージア語	ka
ドイツ語	de
ギリシャ語	el
グリーンランド語	kl

グアラニー語	gn
グジャラト語	gu
ハウサ語	ha
ヘブライ語	he
ヒンディー語	hi
ハンガリー語	hu
アイスランド語	is
インドネシア語	id
インターリングア（国際語）	ia
インターリング語	ie
イヌクティトット語	iu
イヌピア語	ik
アイルランド語	ga
イタリア語	it
日本語	ja
ジャワ語	jv
カンナダ語	kn
カシミール語	ks
カザフ語	kk
キヤーワンダ語（ルアンダ語）	rw
キルギス語	ky
キルンディ語（ルンディ語）	rn
韓国語	ko
クルド語	ku
ラオタ語	lo
ラテン語	la
ラトビア語（レット語）	lv
リンブルク語	li
リンガラ語	ln
リトアニア語	lt
マカドニア語	mk
マダガスカル語	mg
マレー語	ms
マラヤーラム語	ml
マルタ語	mt
マオリ語	mi
マラッタ語	mr
モルダビア語	mo
モンゴル語	mn
ナウル語	na
ネパール語	ne
ノルウェー語	no
オキタン語	oc
オーリア語	or
オロモ語（ガラ語）	om
パシト語（パシュトー語）	ps
ポーランド語	pl
ポルトガル語	pt

パンジャビ語	pa
クエチュア語	qu
レートロマンス語	rm
ルーマニア語	ro
ロシア語	ru
サモア語	sm
サングホ語	sg
サンスクリット語	sa
セルビア語	sr
セルボクロアチア語	sh
セト語	st
セツワナ語	tn
ショナ語	sn
シンド語	sd
シンハラ語	si
シスワティ語	ss
スロヴァキア語	sk
スロヴェニア語	sl
ソマリ語	so
スペイン語	es
スーダン語	su
スワヒリ語（キスワヒリ語）	sw
スウェーデン語	sv
タガログ語	tl
タジク語	tg
タミル語	ta
タタール語	tt
テルグ語	te
タイ語	th
チベット語	bo
チグリニャ語	ti
トンガ語	to
ヅォンガ語	ts
トルコ語	tr
トルクメン語	tk
トウィ語	tw
ウイグル語	ug
ウクライナ語	uk
ウルドゥー語	ur
ウズベク語	uz
ベトナム語	vi
ヴォラピュック語	vo
ウェールズ語	cy
ウォロフ語	wo
コーサ語	xh
イディッシュ語	yi
ヨルバ語	yo
ズールー語	zu

１１－２．言語依存部分の外部テキスト化

（1）ディレクトリ構成

複数言語で切換えながら操作できる環境が実現できるように、セットアップにおいて多言語対応のディレクトリを新たに設けることとした。このディレクトリの位置は自由に設定することができ、環境設定ファイル kdbms.set の LANG_PATH エントリによって定義する。デフォルトでは、ホームディレトリ／ksim／Language である。

各言語に対応した、テキストデータは、この下に、表１１－１で示したコード名のサブ

ディレクトリを設け、そこに格納する。

[例]日本語関連：c:¥@keikan¥ksim¥Language¥ja

（2）リソース

メニュー項目や、操作ボタンの表記等は、従来のシステムにおいては、リソース中に記述され、実行形式の中に固定的に埋め込まれていたが、多言語版においては、テキスト形式外部ファイルとして動的にロードすることとした。

各メニュー項目や、ダイアログのコントロールに表示する、各言語による表記（文字列）は、xml ファイルによって定義している。この xml ファイルは、ksim/bin ディレクトリに格納されたリソースファイルおよびヘッダファイルから、sim.exe が自動的に生成したものを出発点として、翻訳家によるテキスト・ベースの作業を行うことにより完成する。

（3）プログラム中使用する固定的文字列

従来プログラム中に埋め込まれる文字列は、直接プログラム中に記述するのではなく、ソースコードの kj_sjis.h に集約し、他言語に翻訳移植する場合には、このヘッダファイルの翻訳を行った上で、ビルドに使用していた。これは翻訳移植作業を容易化してはいたが、最終的に言語に依存した固定的な文字列は実行形式(sim.exe)の中に埋め込まれるため、修正や他言語への移植に際しては、再ビルドを必要としていた。

各国語専用のプログラム中の文字列はソースコードの kj_sjis.h で定義していたが、これを多言語版においては、LangConv.[言語コード].txt に移し、各言語のディレクトリに置いている。プログラム中で使用される文字列を、システム起動時および言語切換時に動的にロードし、変換テーブル g_TextConvMap により管理し、各種処理に使用する。

ソースコード中においてこの文字列を使用する箇所では、従来の KANJI_XX 等のシンボルは、LangConvert(“KANJI_XX”)という関数に置き換えた。この LangConvert 関数は、上記変換テーブルを用いて、必要な言語による文字列への変換を行う。この変換テーブルは、CMultiLang::InitInfo()関数の中で初期化し構築する。InitInfo 関数は、起動時、及び言語切換時に呼び出される。

（4）ヘルプ・ファイル

ヘルプ・ファイルは、Language/[言語コード]/[ヘルプ名].[言語コード].txt として、用意している。異なる言語のヘルプは、異なるディレクトリに置き、ヘルプが要求された時点で、使用言語に対応したディレクトリ中のヘルプ・ファイルを表示する。

（5）エラー・メッセージ

従来は、ksim/bin/ERR_MSG.txt に格納し、システムの初期化段階でロードしエラーの表示を行っていた。これも、言語毎のディレクトリに格納し、使用言語により選択的にロードすることとした。

（6）その他のテキストファイル

この他に、テクスチャの自動貼り付けのファイルをリストする kdb/texture/sgi/autotex.set、およびプラグイン dll の一覧表である ksim/bin/plugin.tab に

関しても、各言語のディレクトリに翻訳結果が存在すれば、そちらを表示に使用する。いずれも翻訳結果のファイルは、元のディレクトリではなく、Language.[言語コード]のディレクトリに置く。また、ファイル名称も、autotex.[言語コード].set、plugin.[言語コード].set等とし、誤って異なる言語のファイルに上書きしないように配慮している。

なお、この他に、ext.tab (外部ファイル一覧)、roadsec.set、riversec.set（それぞれ道路断面、河川断面を記述したファイル名を格納）等があるが、ファイル名を直接記述し、表示するだけなので、通常は多言語対応する必要はない。もし登録ファイル名に漢字名称等が用いられていて、他国語で表示不能であれば、断面ファイル等の名称を変更した上で、これらのファイルにその名称を直接登録し直す必要がある。

１１－３．表示に使用するフォント

ダイアログの各コントロールに使用されるフォントは、xml ファイルの中で指定することができる。

各ダイアログの定義の中に、例えば

<FONT size=”12” original=”System” replace=”MS UI Gothic” />

という記述で、サイズとフォント名称を指定することができる。

サイズは、xml ファイルを自動生成する際に、リソースファイルで指定されたサイズが初期値として設定されている。

翻訳移植にあたっては、使用する言語が表示できるフォントを指定しなければならない。ダイアログ単位で、サイズと、xml ファイルの中で置換後のフォントを書き換えることにより、特記指定することができるので、ダイアログに使用したリソースの中に表示文字列が納まり切らない場合に、フォントを小さくするという対応も可能である。

実際のフォント生成にあたっては、各ダイアログの初期化の中で、

CMultLang::CMultiLangDialogConv(変換後言語名称, 変換対象ダイアログ)

が起動され、この中でまず、指定されたサイズの論理フォントを作成し、各コントロール（ボタン、固定的テキスト表示、チェックボックス、ラジオボタン等々）の表示書き換えが実行される際に、このフォントを使用する。

フォントの生成には、CFont::CreatePointFont(size, name) 関数を用いて、サイズとフォントの種類だけを指定する方法を採っている。ここで生成したフォントが、不要となるまでの間、固定的なアドレスに保持されていなければ、再描画の際にシステム・フォントが使用されてしまう。そこで、CMultilang::FontManager 関数により、数多くのダイアログに使用される論理フォントを、サイズと名称でソートした上で、スタティックな配列に割り付ける形で管理している。この論理フォントのデータは、言語の切り替えに際して再構築され、システム終了時に除却される。各コントロールへのフォントと表示文字の設定を行う場所は、ダイアログにより異なるが、各クラスの構築の中で行っている場合、OnInitDialog 関数、OnCreate 関数の中で実行している場合などがある。

フォントを管理するためのメモリ・ブロックを解放するためには、各ダイアログの終了時点で、ハンドラのクラスのデストラクタ関数等において、

FontManager(0,"");

を実行する。FontManager 関数は、この引数を見て、メモリを解放する。

１１－４．ダイアログのレイアウト

使用する言語によって、ダイアログを構成するボタンやエディットボックスのそれぞれについて納まりの良いサイズやレイアウトは変わる。そこで、現在は、リソースの中に、日中韓台などの２バイト系文字によるレイアウト（横幅小に対応した IDD_DIALOGXXX と、アルファベット等の１バイト系文字によるレイアウト（横幅大）に対応した IDD_DIALOGXXX_0 の２種類を用意し、指定した言語が１バイト系か２バイト系かで、使用するリソースを使い分けている。

各リソースの中の、言語に依存する文字列に関しては、１バイト系しか表示することのできない OS 環境における環境設定（適切にインストールされていない場合の表示など）に資するため、全て１バイト系のコードが用いられている。実行形式が起動してから、必要とする言語の文字列に動的に置き換えられる。

ダイアログテンプレートを選択するためには、各ダイアログのハンドラ・クラスのコンストラクタ関数の中で、

m_lpszTemplateName = MAKEINTRESOURCE("ダイアログ ID");

を実行すれば良い。

別の方法として、ダイアログを別のクラスからモードレス・ダイアログとして開く場合には、Create 関数の引数として、明示的にダイアログテンプレートを引数として指定する方法がある。

モーダル・ダイアログの場合には、DoModal で起動する際には、Create 関数が自動的には呼び出されないため、前者の方法が有効である。

１１－５．言語の切換操作

以上により、一つのインストール中に複数言語を共存させ、必要な言語に切り替えることが可能となった。

言語の切り替えは、新たに追加したメニューの[ファイル][言語選択]により起動するダイアログで行う。このメニューに一覧表示される言語は、Language ディレクトリにある言語毎のサブディレクトリ名である。

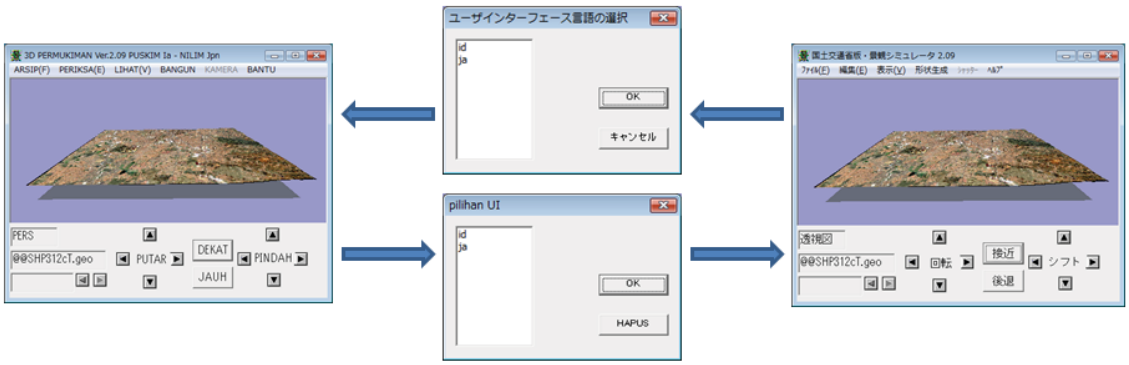

図１１－１：言語切換ダイアログ

１１－６．動作の詳細と、処理プログラム

新たに追加されたソースコードは、language.cpp の中に集約されているが、これに付随して、UNICODE 変換機能や、XML ファイル解析機能を担う補助的な以下のソースコードも追加されている。

表１１－２：多言語機能のための関数を含むソースコード

１．主要部分 multilang.cpp, multilang.h ２．XML ファイル解析機能（オープン・ソースを利用） tinystr.cpp, tinystr.h tinyxml.cpp, tinyxml.h tinyxmlerror.cpp tinyxmlparser.cpp ３．SJIS, utf-8 及び utf-16 の間の２バイト文字コード変換 UnicodeF.cpp, UnicodeF.h ４．外部関数およびプラグイン DLL における多言語対応機能 LocalLang.cpp, LocalLang.h これは、LocalLang.lib として、外部関数、プラグイン DLL から参照する。

さらに、各ダイアログをコントロールする処理の中にも、言語切り替えの処理が追加されている。多言語化のために追加された部分に関しては、表１１－３のようにプリプロセッサで囲ってあるため、多言語版のビルド環境においては、プリプロセッサ MULTI_LANG を定義しておく必要がある。

表１１－３：多言語化のために追加した部分の表記

```
#ifdef MULTI_LANG
  (追加されたソースコード)
#endif
```

（1）初期化動作

言語が設定された時点で、実行形式と同じksim¥binディレクトリにlang.ctlファイル が作成される。これには言語の種類と、言語関連のテキストファイルを格納する Languageディレクトリのパスが記録されている。起動した時点で、lang.ctl が既に存在していれば、初期表示言語は、lang.ctl ファイルに記録されている言語に設定される。動作中に言語が変更された場合には、選択された言語をもってlang.ctl ファイルを上書きする。

Sim.exe が起動した時点で、CSimApp::CSimApp() から、CMultiLang::CMultiLang()が呼び出され、この中で、Lang.ctl に記憶された前回使用言語に基づく初期化動作を行う。Lang.ctl ファイルが存在しなかった場合には、OS が日本語を表示可能である場合には、日本語、そうでない場合には、インドネシア語（アルファベット）による初期表示を行う。初期化時点での言語は、m_Lang に２バイトの文字列として格納され、以後の処理に用いられる。言語が変更された時点で、使用されていた言語がLang.ctl に記録される。

lang.ctl ファイルに記録する、言語を示すコードは、表１１－１に示した通り、ISO 639 に従い、id（インドネシア語）, ja（日本語） などのアルファベット２文字である。

多言語版においては、ksim¥bin ディレクトリの中に、従来からの実行形式本体であるsim.exe に加えて、開発環境からコピーした、リソースの文字情報を含む sim.rc およびresource.h（ビルドの中に含まれるリソース・ファイル）の二つのファイルを追加で置く。起動した sim.exe は、設定された言語を取得した後に、sim.rc と、これを展開し言語依存部分のテキストを格納した、SimRc.[言語コード].xml というｘｍｌ形式で訳語を格納したファイルの有無の確認と、タイムスタンプを比較する（図１１－２）。

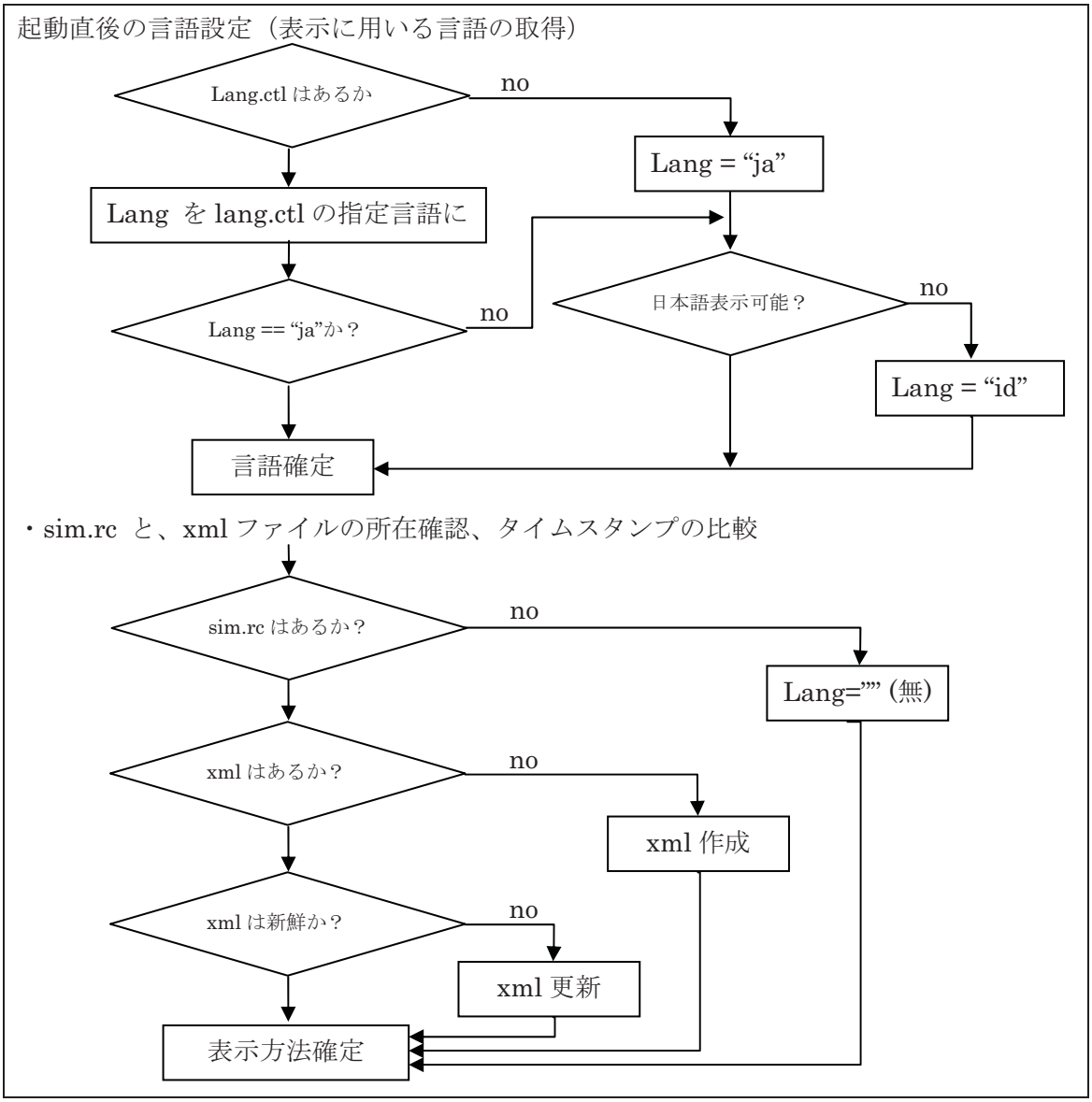

図１１－２：sim.exe の、多言語に対応した初期化フロー図

もし xml ファイルが存在し、かつ sim.rc よりも新しければ、これをそのまま使用して、各ダイアログのための表記文字をロードする。また、もし xml ファイルが存在しなければ、新たに訳語が未記入の新しい xml ファイルを作成し、これを表示に用いる。

一方、既存の xml のタイムスタンプの方が、sim.rc よりも古ければ、この xml が作成された後に sim.exe のバージョンの変更（デバッグ、機能追加等）が行われ、xml ファイルは既に陳腐化していると判定し、新たな xml ファイルを作成する。その際に、旧版の xml ファイル中の訳語が入った項目を保ちながら、新たに追加されたメニュー項目があれば、対応するエントリーだけを作成し、訳語は空欄として、翻訳者による補完に備える。

実行段階では、ロードした xml ファイルの中に、表示に必要な訳語が存在しない場合（空白文字列：訳語未記入）には、リソースファイルの ID コード（“IDD_DLG_XXX” 等)に対

応する、xml で「id=”文字列”」で定義された文字列をアルファベットで表示に用いることで、翻訳が必要な項目が存在することをユーザーに知らしめる。

xml ファイルが使用できない場合（Lang=””（言語無指定））には、ビルド時にリソースで定義された表示文字(もし xml ファイルがあれば、original=”文字列”で定義された文字列に対応する)で表示を行う。

（２）各ダイアログの初期化

各ダイアログのタイトル、メニュー、ボタン等を表記する言語は、各ダイアログの初期化段階で設定される。比較的単純な経年変化ダイアログの場合で見ると、

CDispkeinen::CDispkeinen(CWnd: pParent) 関数の中で、使用する言語が２バイト系か１バイト系かを判定した上で、

Create(IDD_DLG_KEINEN_HENKA, NULL) あるいは

Create(IDD_DLG_KEINEN_HENKA_0, NULL) を実行し、さらに

theApp.m_pLang->MultiLangDialogConv(“IDD_DLG_KEINEN_HENKA”,this)又は

theApp.m_pLang->MultiLangDialogConv(“IDD_DLG_KEINEN_HENKA_0”,this)を実行する。後者の処理の中で、現在選択されている言語による表記が各要素に対して設定される。

DoModal 型のダイアログの場合、Create 関数が呼び出されずにダイアログが開く。また、予め Create を行っていると、DoModal の時点でエラーを生じる。そこで、構築の中で、使用するダイアログテンプレートを示す CDialog のメンバ変数を書き換えておく。

m_lpszTemplateName = MAKEINTRESOURCE(IDD_DLGXXX_0)

この変数は、通常は、ヘッダーの中で、ダイアログのメンバ変数 IDD = IDD_DLGXXX（漢字系）として初期化されている。この方法は、Create 関数を用いる場合にも適用することができる。

メイン画面からユーザーにより起動される各ダイアログが既に開いている状態において言語の切り替えはできない。

唯一、メイン画面のダイアログに関しては、言語の切り替えのみならず、言語が切り替えられた時点で、動的にダイアログを作り直す処理を行っている。

（３）言語切り替え動作

言語切り替えは、メインの[ファイル][言語の選択]により、選択可能な言語の一覧を表示し、そこからユーザーが選択することにより実行する。

CDlgSelLang という新たに追加されたクラスが処理を行う。この中で選択可能な言語がリストボックスで表示される。言語のリストは、Language ディレクトリの中にある、サブディレクトリ名の一覧となっている。この中から言語が選択され、OK により、言語切り替え処理に進む。

切換処理の中では、メイン画面以外のすべてのダイアログを閉じる。メモリ上にロードされているテキスト関連のデータ（ダイアログ、固定的文字列その他）を解放し、新たに

選択された言語のデータに入れ替える。次にこれを用いて、メイン画面のメニューと、下部の操作ボタン等の表示を、新たな言語に切り替える。この時、ロードされていたデータはそのまま継承され、グラフィックな表示は変わらない。

各編集等のダイアログは、次に構築・表示される際に、新たに設定された言語で表示される。

（4）ヘルプ処理

ヘルプは、テキストを notepad.exe または textviewer.exe によって表示するという方法を採用した。考え方として、ユーザーが発見したことをメモとして追加できることを目指した。表示に用いられるテキスト・エディタまたはテキスト・ビューワに、引数としてテキストファイル名を添えて、プロセスを起動することにより実現している。

ファイル名称は、概ね、各種ダイアログのプログラム上の名称に対応した名前に、「.txt」を拡張子として有している。従来のヘルプ・ファイルは、環境設定ファイルの、HELP_PATH エントリで定義されたディレクトリ（デフォルト設定は、ksim/help）に置かれている。多言語対応版においては、Language ディレクトリの下の言語別のディレクトリに置かれる。

更に、多言語化したために、ヘルプ・ファイルは言語毎に別のファイルとして作成される。言語を識別するために、「[固有名称].[言語コード].txt」という名称としている。日本語が選択されている場合に、言語別のディレクトリに、多言語版のヘルプ・ファイルが存在していない場合には、従来のヘルプ・ファイルを表示する。

引数として渡されるヘルプ・ファイルを、使用言語に対応して切り替えるために、Ver.2.07 までは、Pembantu()関数を使用して、CreateProcess 関数で notepad.exe を起動していた。多言語化を実現するために、これに代わって新たに PembantuX()関数を用意した。この関数は、現在選択されている言語を用いて、ヘルプ・ファイルが存在するディレクトリと、ヘルプ・ファイルの名称を作成した上で、コンテンツの表示を行う。ヘルプ・ファイルが存在しない場合には、新たに作成する。

最後に探しに行ったディレクトリにファイルが存在しなかった場合には、Pembantu 関数は、空白の状態で notepad.exe を起動する。ここでユーザーが適切な入力を行い、ファイル保存すれば、以後同じ状況でこのファイルがヘルプ・ファイルとして使用される。既存のファイルが表示された場合も、ユーザーが内容を追記して上書き保存した上で終了すれば、修正されたファイルが以後のヘルプ表示に使用される。

なお、Pembantu 関数は、ファイル名がフルパスで指定されていた場合には、ヘルプ・ファイルが格納されているディレクトリ以外の場所にあるテキストファイルを表示する。ヘルプ以外の局面においても使用される場合がある。例えば、報告書作成機能においては、現在編集中の地物のグループ総数、表示面総数、参照外部ファイル、適用しているマテリアルやテクスチャ等を集計した上で TEMP ライブラリにテキストファイルとして保存した上で、Pembantu 関数を用いて表示を行う。また、平面編集機能においては、現在編集中の面の頂点座標を出力し、Pembantu 関数で表示する機能を用いている。

現時点で実装されているヘルプ・ファイルの一覧を、リスト１１－４に示す。

リスト１１－１：ヘルプ・ファイル一覧

```
1997/09/01  14:55               419 Antiarea.txt
1997/09/01  14:55               292 dispkein.txt
1998/05/16  10:13             1,396 Editligh.txt
1997/10/15  00:27               775 Editmate.txt
1998/11/13  09:10             1,054 Editmove.txt
2003/05/31  08:53             1,314 Editshit.txt
1998/05/09  08:34             4,234 Edittext.txt
1997/09/01  14:54               163 Elembrid.txt
1997/09/01  14:54                75 Elemgras.txt
1997/09/01  14:54               213 Elemstee.txt
1997/09/01  14:53               859 Extwnd.txt
1998/04/21  18:11               283 Grid.txt
2003/10/02  09:44             3,107 Gydlg.txt
1997/09/01  14:53               358 Haichdt.txt
1999/01/09  11:38             5,853 Haichi.txt
1997/09/01  14:52               391 Haichipr.txt
2008/09/27  23:14               783 hyoukou.txt
1997/09/01  14:52                61 Imadlg.txt
2003/10/26  23:54                82 IpErrLog.txt
2008/09/27  18:50               623 kabe.txt
1997/09/01  14:52               577 Kashiwnd.txt
1997/09/01  14:52                91 Keirownd.txt
1997/09/01  14:52               338 Kougen.txt
2003/11/07  17:03             7,180 Koumain.txt
2008/09/28  05:47               132 land.txt
1999/04/20  10:42             5,913 mainfrm.txt
1997/09/01  14:50               118 Mojidlg.txt
1997/09/01  14:50               114 Noridlg.txt
1997/09/01  14:49               498 Noriwnd.txt
2005/10/26  04:32               404 opedlg.txt
1998/04/22  19:01             1,531 Planewnd.txt
2008/09/28  06:12               392 pmove.txt
1998/12/22  14:25               348 pnts.dat
1997/09/01  14:49                87 Primcily.txt
```

```
1997/09/01   14:49              454 Primcone.txt
1997/09/01   14:49              449 Primcube.txt
1997/09/01   14:49              281 Primflco.txt
1997/09/01   14:48              137 Primflcy.txt
1997/10/15   01:01              433 Primline.txt
1997/09/01   14:48              162 Primsphe.txt
1997/09/01   14:48              889 Roadwnd.txt
1997/09/01   14:48              150 Savedlg.txt
1997/09/01   14:47              708 Shitenwn.txt
1997/09/01   14:47              331 Shutterd.txt
1997/09/10   14:35              613 sweep1.txt
1997/09/17   08:23              829 sweep2.txt
2008/09/27   23:23              228 topocut.txt
2008/09/27   23:40              772 topoedit.txt
2008/09/30   07:22            1,374 tunnel.txt
1997/09/01   14:47              341 what.txt
2003/08/28   16:28            6,949 YuuMain.txt
2003/08/28   17:41            5,598 Zaimain.txt
```

（5）プログラム中の文字列定数

景観シミュレータのプログラム中に用いる文字列定数は、初期の段階では、グラフィック・ワークステーション(EUC 系コード)と PC（Shift-JIS 系コード）のマルチ・プラットフォームの要請と、外国語に翻訳する必要性から、ソースコード中の文字列定数を、KANJI.h というヘッダファイルで定義し、その中で

```
#define KANJIxx  文字列
```

という形式でラベルに変換し、このラベルをプログラム中で使用してきた。しかし、この方法では、動的に言語を変換することができないため、多言語に対応した Ver.2.09 においては、LangConvert()関数に引数として KANJIxx を渡すことで、動的に、すなわちその時点で選択されている言語における KANJIxx に対応する文字列を取得している。

この文字列データを格納するファイルは、LangConv.yy.txt というファイル名で、言語毎のディレクトリに置く。言語毎のファイルは、

```
KANJI_xx  “文字列”
```

という形式で変換テーブルを定義する。

１１－７．外部関数及びプラグイン DLL の表示言語の協調動作

ユーザーが、外部関数、及びプラグイン DLL を多言語対応で起動する場合には、以下の

条件が整っていれば、基幹部分で選択された言語を外部関数及びプラグイン DLL で協調的に利用できる。以下、外部関数またはプラグインの名称を XXX、言語コードを YY、ヘルプを必要とするダイアログ名称を ZZZ とした時に、

①外部関数またはプラグイン DLL のビルドに使用されたリソースとヘッダーのコピーが、それぞれ XXX.rc、XXX.rc.h という名称で、ksim/bin ディレクトリに存在する。

②リソースに基づいて構築され、翻訳結果を追記した Language/YY/XXX.YY.xml が、言語毎のディレクトリに作成されている。

③上記②の xml ファイルの末尾に Kanji エントリーが作成されている（プログラム中で使用する言語依存の文字列)。

④外部関数の場合、ヘルプを記述した Language/YY/XXX.YY.txt　が作成されている。

ただし、旧バージョンのインストール環境との互換のため、ヘルプの ja（日本語） に限って、従来の Ksim/Help にあっても正常に動作する。プラグイン DLL の場合には、基幹部分のヘルプ表示機能が利用できるため、ファイル名は、任意に設定することができ、通常、基幹部分と同様 Language/YY/ZZZ.YY.txt という名称となる。

外部関数、及びプラグイン DLL は、起動時点で bin/Lang.ctl を開き、基幹部分が現在使用している言語の種類を知る。この内、①から②を作成する機能に関しては、プログラミングを容易化するために、共通の機能として、sim.exe 本体の側で、外部関数及びプラグイン DLL を起動する時点で、xml ファイルを外部関数またはプラグイン DLL のリソースから新規作成または更新（タイムスタンプ比較による）を行っている。従って、外部関数またはプラグイン DLL の側でリソースを確認し xml ファイルを作成する必要はない。

外部関数側で共通の②③④のファイルを利用する機能に関しては、LocalLang.cpp でコーディングし、LocalLang.lib というスタティック・ライブラリに集約してあるため、開発にあたっては、これをリンクに加えると共に、必要な部分でライブラリ関数を実行することにより、多言語に対応した外部関数、プラグイン DLL を開発することができる。

１１－８．外部関数及びプラグイン DLL における多言語機能の実装方法

（1）多言語版の外部関数、及びプラグイン DLL の作成方法

外部関数を多言語で使用するためには、メニュー項目、ダイアログの各コントロール（ボタン、チェックボックス、固定的文字表示など)、ヘルプ、メッセージ等の表示を、複数言語の中から現在選択されている言語に切り替えて表示できる条件を実現する必要がある。更に、言語により必要なボタンサイズの横幅が異なるため、漢字系とアルファベット系の二つのレイアウトを用意する必要がある。

実行段階では、メイン側で、外部関数のリソースファイルを解析し、外部関数が言語の切り替えに必要とする情報を格納した言語別の XML ファイルを構築してから外部関数を起動する。従って、外部関数の側では、本体から指定された言語の XML ファイルを用いて、

ダイアログを構築する処理だけを行えば良い。

外部関数を開発するプログラマにより、少なくとも漢字系とアルファベット系の２種類の表示レイアウト（リソースファイルで定義されたダイアログテンプレート）が実現されていれば、後はプログラミング作業なしに、翻訳者によるテキストファイルの作成のみで、第三の新しい言語への対応が可能となる。

（２）実行環境

要求された言語での表示を行うためには、以下のファイルが、必要なディレクトリに存在することが条件

・ksim/bin ディレクトリに、[関数名]_D.rc というリソースファイルがコピーされていること。

・ksim/bin ディレクトリに、[関数名]_D.rc.h という名前に修正した、リソースのヘッダファイルが存在していること。

なお、開発環境が標準的に生成するリソースのヘッダファイルの名称は一律に、「resource.h」であるため、基幹部分のリソースや他の外部関数、プラグイン DLL 等と区別することができない。そこで、各機能の開発においては、上記の名称に変更した上で、このヘッダファイルを参照する部分にも対応する修正を施した上でビルドを行う。

（３）起動前の準備処理

外部関数を起動する sim.exe の側では、プロセス起動に先立って、以下の準備処理を行う。

① 要求された言語に対応する xml ファイルが存在しない場合には、新たに上記のリソースとヘッダーから作成する。xml ファイルの所在と名称は以下の通り：

[LangPath]¥[言語名]¥[関数名].[言語名].xml

例： ksim¥MultiLang¥ja¥sample.ja.xml

② 存在する場合には、リソースファイルと xml ファイルのタイムスタンプを比較し、リソースのほうが新しければ更新をかける。その際に、新たなダイアログやコントロールが追加されていれば、これに対応する新たなエントリーを作成し、翻訳結果を空白文字列とする。削除されたダイアログやコントロールについてはそのまま残す。

③ ksim/bin ディレクトリに選択された言語を記録した lang.ctl ファイルを作成（更新）する。

このファイルは、sim.exe が次に起動した際の初期表示言語の選択と、外部関数やプラグイン DLL の起動時における表示言語の選択に使用される。

lang.ctl ファイルには、以下の情報が含まれている：

第１行　言語コード　　例：ja

第２行　言語ディレクトリ　　例：c:¥@keikan¥ksim¥Language

（４）外部関数・プラグイン DLL の側でのダイアログ構築処理

起動する言語の側では、以下の処理を行う。

① 起動された実行形式自身が存在するディレクトリに lang.ctl ファイルが存在すれば

これを用いて表示すべき言語を認識し、必要な xml ファイルをロードする。存在しなければ、日本語として処理する。

② xml ファイルの定義に従って、各ダイアログのメニュー、コントロールの表示を切り替える。その際に、言語が２バイト系か１バイト系かを識別して、ダイアログの配置に使用するリソースを選択する。

リソースにおけるダイアログは、２バイト系に関して［ダイアログ名］、１バイト系に関して［ダイアログ名］_0 の名称を付して識別する。

リスト１１－２：プラグイン DLL における多言語化のための追加部分

関連するソースの冒頭部分

```
#ifdef MULTI_LANG
#include "LocalLang.h"
extern CLandApp theApp;
#endif
```

OnCreate（ダイアログのリソースの選択とメニューの設定）

```
#ifdef MULTI_LANG
        if(IsKanjiDlg("land.dll")){
                if(!m_Prm_Bar.Create(this, IDD_LAND_DLG_HYOUKOU,

        CBRS_ALIGN_RIGHT | CBRS_TOOLTIPS | CBRS_FLYBY,

        IDD_LAND_DLG_HYOUKOU)) {
                        TRACE0("m_Prm_haichi_Bar create faild¥n");
                        return -1;
                }
                MultiLangDialogConv("IDD_LAND_DLG_HYOUKOU", (CDialog*)&m_Prm_Bar);
        }
        else{
                if(!m_Prm_Bar.Create(this, IDD_LAND_DLG_HYOUKOU_0,

        CBRS_ALIGN_RIGHT | CBRS_TOOLTIPS | CBRS_FLYBY,

        IDD_LAND_DLG_HYOUKOU_0)) {
                        TRACE0("m_Prm_haichi_Bar create faild¥n");
                        return -1;
                }
                MultiLangDialogConv("IDD_LAND_DLG_HYOUKOU_0", (CDialog*)&m_Prm_Bar);
        }

        if(IsMultLangOK()){
                CMenu* pMenu = GetMenu();
                MultiLangMenuConv(pMenu,"IDR_MENU_LAND_HAICHI", 0, 0);
        }
#endif
```

OnInitDialog（ダイアログの表記文字の張替を行う）

```
BOOL CLandMenu::OnInitDialog(){
        CDialog::OnInitDialog();
        // 初期化の補足処理
        SetIcon(AfxGetApp()->LoadIcon(MAKEINTRESOURCE(IDI_SIM)),1);
#ifdef MULTI_LANG
        if(IsKanjiDlg("land.dll")){
                MultiLangDialogConv("IDD_LAND_DLG",(CDialog*)this);
        }else{
                MultiLangDialogConv("IDD_LAND_DLG_0",(CDialog*)this);
        }
#endif
        return TRUE;
}
```

終了処理

```
#ifdef MULTI_LANG
        MultiLangEnd();
#endif
```

（５）外部関数及びプラグイン DLL における文字列定数

選択的にセットアップされるプラグイン DLL と外部関数共に、翻訳を必要とする文字列は、本体が用いているデータに加えることができないため、独自のデータとして用意する必要がある。そこで、言語別 xml ファイルのルート・ノードとして、<Kanji>タグを使用し、このタグの中に、個々のプラグイン DLL および外部関数が使用する文字列定数を格納する方法を LocalLang ライブラリが提供している。

LocalLang.lib の中の、Kanji(“KANJIxx”)関数を用いて、現在選択されている言語での文字列を取り出すことができる。

ComboBox に初期条件としてセットする文字列は、単一言語であれば、リソースファイルの中に格納することも可能であるが、選択された文字列をマッチングすることも考えると、文字列を外部化し、明示的な初期化ルーチンの中で、文字列の初期値をセットする方法が便利である。その際に使用する初期化用、及び検証用の文字列を、文字列定数として、xml ファイルの<Kanji>タグの中で定義することができる。

エラー・メッセージの処理は、外部関数とプラグイン DLL で異なっている。プラグイン DLL の場合には、本体(sim.exe)からエクスポートされるｚ３ライブラリ関数を利用することができる。従って、本体側で用意しているエラー・メッセージ集で提供されているメッセージであれば、これを利用することができる。それ以外の文字列定数は独自に用意しなければならない。

外部関数の場合には、本体とは独立したビルドとなるため、多言語で表示するエラー・メッセージを全て独自にプログラムしなければならない。

そこで、外部関数及びプラグイン DLL の多言語処理のために用意した LocalLang.lib ライブラリでは、ダイアログやメニューを XML から構築する機能を中心とする諸関数に加えて、文字列定数を言語別に利用するための Kanji()関数が用意してある。本体側が自動生成し、これに翻訳家が訳を追記する言語別の XML ファイルの末尾(<SimResource>タグの内側)に、<Kanji>・・・<Kanji/>タグを設け、この中に、

リスト１１－３：Kanji()関数のためのデータの形式

```
<Kanji>
  <KANJI_1 replace=”訳語１”/>
  <KANJI_2 replace=”訳語２”/>
     ・・・・・・・・・・
</Kanji>
```

という形で文字列定数を定義しておくことにより、MessageBox（）関数等を用いて表示する文字列を、言語別に定義することができる。LocalLang.lib の Kanji()関数の引数として上記の「KANJI_1」などのタグを文字列として渡すことにより、現在選択されている言語の XML ファイルにおけるこのタグ中の「replace=」で定義された文字列が返される。これを用いることにより、エラー・メッセージや、コンボボックスの初期化、文字列比較などの処理を行うことができる。

（６）外部関数及びプラグイン DLL におけるヘルプの作成方法の実際

プラグイン DLL においては、基幹部分の関数をインポートすることにより利用できるため、基幹部分と同様の方法で、ヘルプを表示することができる。

外部関数の場合には、本体側の Pembantu、PembantuX 関数が使用できないため、ヘルプ表示を行う機能を独自に作りこまなければならない。特に、ヘルプ・ファイルを格納するディレクトリを環境変数から取得するためには、ＥＮＶライブラリを使用する必要があり、これをリンクすると、小さい単純なダイアログであっても不必要に大きくなる。そこで、本体側から使用言語に関する情報を提供する Lang.ctl ファイルの２行目に、翻訳結果を格納した xml ファイルの所在をフルパスで記述している。この文字列から、ヘルプ・ファイル等の所在を得る方法が、簡便である。このための関数を、LocalLang.lib ライブラリの中に用意した。即ち、LocalLang ライブラリが提供している PembantuX(テキストファイル名)関数を用いることにより、言語別のディレクトリに格納してあるヘルプ表示用のテキストファイルを選択的に表示することができる。

例えば、日本語が選択された状態で PembantuX(“yyy.txt”)を起動すると、言語別のディレクトリ ja にある、yyy.ja.txt というテキストファイルを表示する。

また、より簡便に、引数なしで Bantu()関数を起動することにより、外部関数名またはプラグイン DLL 名と同じ名称を有する言語別のテキストファイルを表示することができる。

例えば、Cube_D.exe 関数の中で韓国語が選択された状態で Bantu()関数(引数なし)を起動すると、言語別のディレクトリ ko の下にある、Cube.ko.txt というファイルを表示する。

（7）外部関数・プラグイン DLL の多言語対応におけるプログラマの仕事

開発する新しい外部関数の例として[例] newfunc のダイアログ部を newfunc_D.exe とする。ダイアログは、外部関数の引数パラメータを入力し、これを実際に形状生成する newfunc.exe に渡すための小さなファイルを作成することが仕事である。

プラグイン DLL の場合には、[例] newdialog.dll とする。プラグイン DLL は、ダイアログから形状生成、表示を含む、全ての機能を実装することができる。ダイアログも複数もつことができる。

当初から多言語版として外部関数を開発する場合には、以下の手順に従う。

①プログラミングに先立って、リソースのためのヘッダーを、最初にデフォルトの resource.h から変更して、newfunc_D.rc.h としておくと便利である。 リソースファイルと、ダイアログ・ハンドラの冒頭部分の#include も、これに合わせて修正する。

②２バイト系のダイアログを日本語表示でデザイン・レイアウトする。

プログラムをコーディングし（ダイアログが表示できれば可）、実行形式 newfunc_D.exe を作る。日本語だけを用いて表示テストを行う場合には、プリプロセッサ MULTI_LANG を定義しない状態でビルドを通す。

③EXT.TAB 関数に、newfunc を登録する。

④Newfunc_D.exe およびリソースとヘッダーを、ksim/bin ディレクトリにコピーし、これらを用いて、日本語表示モードで関数を sim.exe から起動する。この段階で、関数の本来の機能をデバッグする。

⑤この時 sim.exe が、Language/jp ディレクトリに生成する newfunc.ja.xml ファイルを、別フォルダまたは別名称で保存しておく。日本語表記部分を、翻訳結果としてあとで利用する。

⑥リソース・エディタで、２バイト系のダイアログの表記を日本語からローマ字に修正する（この時、ボタン等からはみ出しても気にしない）。

２バイト系のリソースを日本語としたままの場合、日本語が表示できない環境での言語切り替え処理に支障が生じる可能性がある（文字列比較などでエラーを生じ当該言語での翻訳結果に正常に切り替わらない）。また、日本語を出発点として翻訳できない場合に、翻訳家の作業が困難となる。

⑦ newfunc_D.rc を、テキスト・エディタで開き、各ダイアログをコピーし、コピーしたダイアログの名称に「_0」を追加して、１バイト系の表示用ダイアログとする。

リスト１１－４：リソースの中のダイアログのデザイン定義の編集例

編集例：

リソース・エディタで作成したダイアログの配置と構成

```
IDD_DIALOG_SEL_LANG DIALOGEX 0, 0, 167, 94
STYLE DS_SETFONT | DS_MODALFRAME | DS_FIXEDSYS | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "USER INTERFACE GENGO SENTAKU"
FONT 8, "MS Shell Dlg", 400, 0, 0x1
BEGIN
```

```
    DEFPUSHBUTTON    "OK",IDOK,110,39,50,14
    PUSHBUTTON       "CANCEL",IDCANCEL,110,65,50,14
    LISTBOX                  IDC_LIST_LANG,7,7,56,80,LBS_SORT  |  LBS_NOINTEGRALHEIGHT  |
WS_VSCROLL | WS_TABSTOP
END
```

テキスト・エディタで、リソースファイル中に作成したコピー（１バイト系言語用）

```
IDD_DIALOG_SEL_LANG_0 DIALOGEX 0, 0, 177, 94
STYLE  DS_SETFONT  |  DS_MODALFRAME  |  DS_FIXEDSYS  |  WS_POPUP  |  WS_CAPTION  |
WS_SYSMENU
CAPTION "USER INTERFACE GENGO SENTAKU"
FONT 8, "MS Shell Dlg", 400, 0, 0x1
BEGIN
    DEFPUSHBUTTON    "OK",IDOK,110,39,60,14
    PUSHBUTTON       "CANCEL",IDCANCEL,110,65,60,14
    LISTBOX                  IDC_LIST_LANG,7,7,56,80,LBS_SORT  |  LBS_NOINTEGRALHEIGHT  |
WS_VSCROLL | WS_TABSTOP
END
```

（ダイアログの ID と、配置、コントロールのサイズを示す画面座標値以外は同じである）

⑧ newfunc_D.rc.h をテキスト・エディタで開き、上記のコピーしたダイアログ名称に、他と重複しない ID 数値を定義する。

⑨グラフィックなリソースエディタ（開発環境）で newfunc_D.rc を開き、１バイト系のダイアログ_0 をローマ字表示でレイアウトする。多くの場合、アルファベット表記の方が横長となるため、ボタン等を横に長くデザインする必要がある。

⑩すでに④⑤で作成してある、newfunc.ja.xml のコピーと、ローマ字表記の newfunc_D.rc を、二つのテキスト・エディタで開き、newfunc.ja.xml の original = "日本語表記" replace = "" というペアを、original ="ローマ字表記" replace = "日本語表記"　となるように修正する。

長い文字列等の場合、replace="" の側にローマ字表記を入力した上で、original= を noriginal= と簡単に修正する。更に、既にある original="日本語" を noriginal="日本語" と簡単に修正しておく。最後に、全文対象置換をかけて noriginal → replace、nreplace → original 等と一括処理すると能率的である。

⑪リソースファイル newfunc_D.rc と、ヘッダファイルを newfunc_D.rc.h として、再度 ksim/bin ディレクトリにコピーする。MULTI_LANG プリプロセッサを定義した状態でビルドした、newfunc_D.exe に更新する。

修正（⑩）した日本語表記の newfunc_D.ja.xml を、Language/ja にコピーする。

⑫sim.exe から、この外部関数を、開発に使用した１バイト系言語と、２バイト系言語で起動する。

⑬日本語が正常に変換表示されることを確認する。

⑭新たに生成された、Language/id/newfunc_D.id.xml に訳語を加える（この例ではインドネシア語）。

⑮生成した二つの xml ファイルの各メニュー項目、ダイアログ、コントロールのエントリーに、最終的に表示したい文字列を入力する。なお、リソース定義と同一の表示で良い場合には省略が可能である。

⑯確認と検査を行う。

以上の作業の後、リソース、ヘッダー、二つの言語の XML ファイルをバックアップした上で、関数を二つの言語で実行し、翻訳結果が正確に表示されることを確認する。

次に、二つの xml ファイルをテキスト・エディタで開く。各ダイアログの最後に、新たなエントリーが追加され、訳語が空欄(replace="")となっていないか確認する。もし存在する場合、翻訳作業漏れ、original="ローマ字表記"のミスタイプ（大文字小文字、余白の数等に注意）等が疑われるため、対応する項目を修正した上で、システムが自動的に追加したエントリーを削除する。修正後、再度検査を行い、XML ファイルに変化が無いことをもって終了とする。リソースのミスタイプを修正する場合には、開発環境にあるリソースと、ksim/bin にコピーしたリソースの両方を更新し、同一とする必要がある。

日本語で既に開発されている外部関数の多言語化を行う場合には、言語切り替えの関数を追加することで多言語化ができる。

①各ダイアログの構築部、InitDialog 関数に修正を加え、言語別にダイアログテンプレート、メニュー、及びダイアログの文字列表示を行うようにする。

②リソース・エディタで各ダイアログのコピーを作成し、ダイアログ名_0 という名称とする。これを、１バイト系（欧文）の文字列にふさわしい寸法に調整する。

③リソースとヘッダーの名称を、多言語対応に修正する。

外部関数名_D.exe の場合には、外部関数名.rc および 外部関数名.rc.h

プラグイン名.dll の場合には、プラグイン名.dll.rc 及びプラグイン名.dll.rc.h

とする。

以後は、新たに多言語用として開発する場合の④以下と同様である。

１バイト系外国語で既に開発されている外部関数を多言語化する場合も、本質的には同様である。日本語の訳語をｘｍｌファイルに追記するためには、日本語が入力・表示できる OS 環境が必要であるが、日本語が表示できない環境であっても、日本語を入力する前段の、訳語がまだ空欄のままのｘｍｌファイルを作成するところまでは実行可能である。

（８）外部関数・プラグイン DLL に関する翻訳家の仕事

例えば、newfunc_D.exe というダイアログの、新たな言語 yy で使用する場合、

①出発点として使いやすい XML ファイルをコピーして newfunc.[新たな言語コード].xml (newfunc.yy.xml)というファイルを作成する。

②各項目について翻訳作業を行い、結果を各エントリーの replace="" の部分に、replace="[言語 yy による翻訳結果]" という形で入力する。

③Langconv.yy.txt、ERR_MSG.yy.txt についても、既存の言語のファイルのコピーを基

に翻訳を行う。

④ヘルプファイル（多数）の翻訳を行い、ファイル名を修正する。

以上の翻訳結果を、対応する言語ディレクトリにコピーする。

プログラムのバージョンアップ、デバッグ等に伴い、新たなコントロールが追加されたり、オリジナルの表示文字列が修正されたりした場合にも、翻訳家による訳語の追加補正作業が必要となる。

オリジナルの表示文字列が修正された場合には、以前の翻訳結果が残っているので、replace="" の形で空欄となっている項目のみ訳語を補えば良い。

（9）外部関数及びプラグイン DLL のインストールの方法

①外部関数

EXT.TAB への登録。これはアルファベットで記述された外部関数名称だけを使用しているため、多言語対応は必要ではない。

Ksim/bin ディレクトリに実行形式と XXX.rc、XXX.rc.h を置く。

Language/yy ディレクトリに、XXX.yy.xml を置く

②プラグイン DLL

PLUGIN.YY.TAB という名称で多言語化したファイルに、各言語に翻訳した機能名称と、実行形式の名称を登録する。このファイルは、メニューに追加する各国語での機能名称と、実行形式の名称を登録する必要があるため、使用する言語に対応する Language/yy ディレクトリに置く。

同じ Language/yy ディレクトリに、XXX.yy.xml を置く。

DLL の本体と、リソース、およびヘッダーを ksim/bin ディレクトリに置く。