# 重度視覚障害者によるJavaプログラミングの学習を支援する統合的環境の構築

（課題番号　17500670）

平成17年度～平成18年度科学研究費補助金

（基盤研究（C））研究成果報告書

平成19年3月

研究代表者 長岡 英司

筑波技術大学 障害者高等教育研究支援センター 教授

# 重度視覚障害者によるJavaプログラミングの学習を支援する統合的環境の構築

10810002

# はしがき

本書は平成17・18年度、科学研究費補助金による研究成果報告書である。

**研究課題**：重度視覚障害者によるJavaプログラミングの学習を支援する統合的環境の構築

課題番号：17500670

**研究組織**：

研究代表者：長岡　英司（筑波技術大学障害者高等教育研究支援センター　教授）

研究分担者：大武　信之（筑波技術大学障害者高等教育研究支援センター　教授）

研究分担者：金堀　利洋（筑波技術大学障害者高等教育研究支援センター　助教授）

交付決定額（配分額）　（金額単位：円）

| | 間接経費 | 直接経費 | 合計 |
|---|---|---|---|
| 平成１７年度 | 0 | 1,600,000 | 1,600,000 |
| 平成１８年度 | 0 | 1,100,000 | 1,100,000 |
| 総計 | 0 | 2,700,000 | 2,700,000 |

**研究発表**：

（１）学会発表等

1) 長岡英司：全盲者による描画・作画の可能性に関する考察－Java言語の活用－、第31回感覚代行シンポジウム（感覚代行研究会・独立行政法人産業技術総合研究所）、pp67-72、2005.12.5（東京・産業技術総合研究所）

2) 長岡英司・山本卓：重度視覚障害者によるJavaプログラミングを支援するソフトウェアの開発、日本特殊教育学会第44回大会発表論文集、p121、2006.9.16（群馬大学）

3) 山本卓・長岡英司・清水豊：重度視覚障害者のJavaプログラミングを支援するシステムの開発、ヒューマンインタフェースシンポジウム2006論文集、pp1021－1024、2006.9.25（岡山・倉敷）

# 目　次

## 本研究の目的

近年、重度の視覚障害者によるプログラミングが行われなくなった。

1990 年代までは、汎用コンピュータやＤＯＳのＰＣの下で、視覚障害者もいくつかの汎用言語を使って盛んにプログラミングを行った。それによって、情報処理分野に新たな職域が開かれたほか、障害に起因する困難や不便を解消するためのソフトウェアを視覚障害者が自らの手で開発するなど、多くの成果が得られた。しかし、その後、Windows の普及でＧＵＩ化が進展したことなどから視覚障害者が利用できるプログラミング環境がなくなり、有効な対策がなされないまま現在に至っている。

ＰＣやインターネットの活用は、視覚障害者に新たな多くの可能性をもたらしている。しかし、自身でプログラミングができないために、より高度な利用法の開拓や利用環境の改善を主体的に行うことが難しい。研究代表者の調査によると、就業している視覚障害者の多くが、ＰＣの利用環境の変化に対応できないことに強い危機感を持っている。

このような状況にあることから、視覚障害者によるプログラミングを再び可能にするための取り組みが必要である。なかでも、プログラミングの実用的なスキルを実践的に習得できる学習機会を提供することが、極めて重要といえる。

本研究は、視覚障害者のための情報処理教育（あるいは、プログラミングの経験などを有する視覚障害者の自学自習）で利用できる「Java プログラミングの学習を支援する実習環境」を構築するために実施した。

## 本研究の成果

Java 言語でのプログラミングを触覚や聴覚による方法で行えるようにする視覚障害者用支援ソフトウェアシステムを開発し、実用化した。

本システムは、次の３種のアプリケーションソフトで構成される。どのアプリケーションも、点字ディスプレイ端末への点字出力を行うほか、各スクリーンリーダでの音声出力がなされるよう設計されている。

1．テキストエディタ：テキストの編集に必要な最低限の機能に加え、コンパイラと連携する機能、カレットの現在位置（行番号やメソッド名）を読み上げる機能、プログラムの構造の概要を表示する機能（アウトライン機能）などを備えている。
2．代替コマンドプロンプト：コマンドライン方式での対話（コマンド等のキー入力と実行結果等の確認）を点字ディスプレイ出力と音声読み上げを介して行えるようにするソフトウェアである。コマンド等の入力を行うための入力ウインドウと、プログラムからの出力等が表示される出力ウインドウとを切り替えながら使用する。
3．コンパイル補助用フロントプロセッサ：Java コンパイラ javac と同様にコマンドラインで操作してコンパイルを行うソフトウェアであるが、エラーが検出されると、その一覧表示ウインドウが開き、そこでエラー表示の一つを選択すると、ソースコードの当該エラー発生箇所に、上記のテキストエディタで直接にアクセスすることができる。

本システムにより、Java でのプログラミング作業を、点字ディスプレイ端末とスクリーンリーダを介して対話的に進められるようになった。

＊　開発成果物のソフトウェア　AiB Tools　は
http://sgry.sakura.ne.jp/aibtools/download.html
から無償でダウンロードできる。

# 重度視覚障害者によるJavaプログラミングを支援するソフトウェアの開発

○長岡　英司　　山本　卓
（筑波技術大学　障害者高等教育研究支援センター）　（電気通信大学　電気通信学研究科）

KEY WORDS: 重度視覚障害、Javaプログラミング、点字ディスプレイ出力

（目的）

PCやインターネットの活用は、視覚障害者に新たな可能性をもたらした。その結果、教育や職業をはじめとする様々な場面で生活の質的向上が実現している。だが、最近のPC利用環境では、視覚を用いずにソフトウェア開発を行う手段がほとんどないため、より高度な利用法の開発や利用環境の改善に視覚障害者自身が主体的に取り組むことが難しい。こうした状況の改善を目的に、Java言語でのプログラミングを、触覚や聴覚による方法で行えるようにするための支援ソフトウェアを開発した。

（背景）

1990年代までは、重度の視覚障害者も汎用コンピュータやDOS PC上でプログラミングを行っていた。それによって、情報処理分野に新たな職域が開かれたほか、障害に起因する困難や不便を解消するためのソフトウェアを視覚障害者が自らの手で開発するなど、多くの成果が得られた。しかし、その後、Windowsの普及でGUI (Graphical User Interface) 化が進展したことなどから、視覚障害者が利用できるプログラミング環境がなくなり、有効な対策がないまま現在に至っている[1]。

（言語の選定）

Javaは、CやC++などの利点を引き継ぎ、同時にそれらの難点のいくつかが解消された、比較的使い易い言語である。また、現在の標準的な基本ソフトであるWindowsの下ではほとんどすべての開発環境がGUI化している中で、Javaの開発環境JDK (Java Development Kit) は、旧来のコマンドライン方式を採っている。これは視覚障害者にとって大きな利点であり、このことが、Javaを視覚障害者によるプログラミング用の言語に選定した理由である。

（開発成果）

これまでに、次の3種のアプリケーションソフトを開発した。どのアプリケーションも、点字ディスプレイ端末への出力を行うほか、各スクリーンリーダで読み上げがなされるよう設計されている。開発は、C#言語とC++言語で行い、点字変換と点字ディスプレイ出力にはニュー・ブレイル・システム株式会社のNBSエンジンを用いている[2]。

（1）テキストエディタ　AccEdit

テキストの編集に必要な最低限の機能に加え、コンパイラと連携する機能、カレットの現在位置（行番号やメソッド名）を読み上げる機能、プログラムの構造の概要を表示する機能（アウトライン機能）などを備えている。編集用のメインウインドウとアウトライン表示ウインドウで構成されている。

（2）代替コマンドプロンプト AccPrompt

コマンドライン方式での対話（コマンド等のキー入力と実行結果等の確認）を点字ディスプレイ出力と音声読み上げを介して行えるようにするソフトウェアである。コマンド等の入力を行うための入力ウインドウと、プログラムからの出力等が表示される出力ウインドウとを切り替えながら使用する。

（3）コンパイル補助用フロントプロセッサ _javac

Javaコンパイラjavacと同様にコマンドラインで操作してコンパイルを行うソフトウェアであるが、エラーが検出されると、その一覧表示ウインドウ（図1）が開き、そこでエラー表示の一つを選択すると、ソースコードの当該エラー発生箇所にエディタ（AccEdit）で直接にアクセスすることができる。

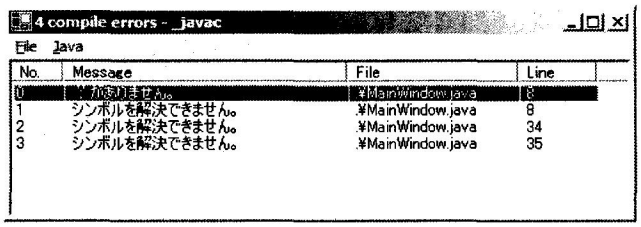



図1:エラーの一覧表示ウィンドウ

これらにより、Javaでのプログラミング作業を、点字ディスプレイ端末を介して対話的に進められる。その流れを図2に示す。

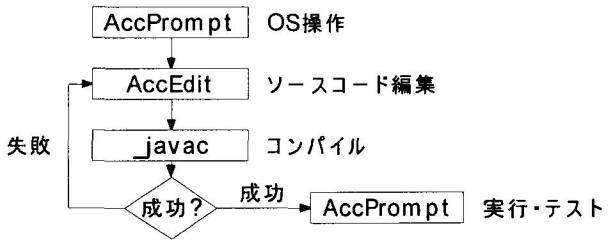



図2:プログラミング作業の流れ

（今後の開発課題）

この方法により、コマンドライン方式のプログラムならば、点字ディスプレイ出力で動作確認をして完成まで至ることができる。だが、GUI方式のプログラムについては、現在のところ、動作を確認する合理的な方法がないため、視覚障害者が独力で完成するのは困難である。いまや、グラフィカルな表示は一般的であることから、今後、この問題を解決する方法を確立する必要がある。

（文献）

[1] 長岡英司（2003）:「重度視覚障害者のソフトウェア開発技能の職業的有用性」,職業リハビリテーションNo.16,pp43-51

[2] 長岡英司、星加恒夫（2004）:「実時間点字表示機能を備えたテキストエディタの開発と活用」,第30回感覚代行シンポジウム,pp99-104

(NAGAOKA　Hideji,　YAMAMOTO Suguru)

この部分は以下の論文で構成されていますが、著作権者（著者、出版社、学会等）の許諾を得ていないため、筑波技術大学では電子化・公開しておりません。

「重度視覚障害者の Java プログラミングを支援するシステムの開発」

ヒューマンインタフェースシンポジウム 2006 論文集, pp1021－1024, 2006

# 全盲者による描画・作画の可能性に関する考察
## — Java 言語の活用 —

長岡　英司
筑波技術大学　障害者高等教育研究支援センター

Abstract:　Graphical expression is useful for the sighted to acquire information. If the blind can send information in graphical expression, their communication with the sighted must be improved. Therefore, a project was planned to develop a practical method which would enable the blind to produce visual graphics for transmitting information. The drawing function of Java language was selected to be used as a tool for describing graphical objects. In the preliminary stage, the author, totally blind, actually tried drawing a picture by means of Java and clarified the possibilities and problems of this method.

Keywords:　Blindness, Graphics, Java, Braille, Tactile Graphic Display

## 1．はじめに

近年、IT(情報技術)の活用で視覚障害者の情報環境は飛躍的に改善された[1]。そのうち、全盲者による情報の発信については、漢字を含む墨字(普通の文字）の文書をPCによって独力で作成できるようになったことが、状況を大きく変えた。それ以前は、晴眼者(視覚に障害のない人）に向けての情報の発信では全面的に助力に依存しなければならなかったが、墨字による自立的な情報発信が可能になって、全盲者の社会参加が促進された。また、インターネットの普及に伴って、全盲者が晴眼者に向けて情報の発信や伝達をする機会と可能性が、さらに増大した。しかし、晴眼者の間では情報を（視覚的に）多彩に表現することが一般的になっている中で、全盲者が発信する情報は文字の連なりだけで表記されたものがほとんどであり、これでは受け手の関心や注目、さらには理解が十分に得られない場合がある。このような状況の改善には、画像の利用や描画を可能にするなど、全盲者が情報を視覚的に表現することに主体的に関われる方法や手段の整備が必要である。そうした可能性を探る一環として、視覚を用いずに簡単な画像を作成する一つの方法を試み、それを有用なものとして確立するための今後の課題を検討した。

## 2．描画や作図のための既存の手段

1960年代までの日本の盲学校教育では、板に蚊帳用の布を貼り付けた「図画板」（通称「蚊帳板」）という用具が、触図の作成や提示に用いられていた。点字用紙をその上におき、点筆やルレットなどの器具で図形等を描くと、用紙の裏側に突点の連なりでそれが浮き出る。表側でも軌跡を触知できるので、全盲者による描画や作図のための道具としても用いられた。しかし、描く側（表）と触読する側（裏）で画像の向きが逆になることなどもあって、触覚のみでの描画や作図は容易でない。

「レーズライター」（表面作図器）は、シリコンゴムの下敷きの上においた特殊な用紙に通常のボールペンで図形等を描く視覚障害者のための筆記・提示用具である。ある程度の筆圧をかけて描くと、ペン先の軌跡が突状に盛り上がって固定し、それを触知できる。1960年代後半に開発されたこの用具[2]は、その後、図画板に取って代わって、盲学校教育などで広く活用されるようになった。これを用いれば、全盲者が独力で図形等を描くことが可能である。しかし、触覚による描画では、正確な図形や複雑な画像を描くことは難しい。

一方、渡辺と小林が開発した「電子レーズライタ」[3]は、PCに接続した点図ディスプレイ上で、ペン型ポインティングデバイスを使って図形等を描くシステムである。ペン先が触れながら通過した位置にあるピンが突出して留まり、それが触覚的なフィードバックとなって、全盲者による描画を可能にしている。消去・修正も自在にできる。しかし、触覚の特性や点図ディスプレイのピンの密度の問題から、正確な描画は困難である。

さらに、藤芳が開発したBPLOT[4]は、重度視

覚障害者が独力で点図を作成するためのシステムである。「座標で指定した位置に線を引く」などの作図コマンドの列で点図を記述し、点字プリンタ・プロッタに印刷させる。

国内で使用もしくは開発されているこれらの手段は、いずれも触知図形を描くのに使われるものであり、視覚的に提示する対象を作成するためのものではない。

## 3．方法開発の必要性

プレゼンテーションを行うことに関する重度視覚障害者の現状を把握するためのアンケート調査を、2004年6月から7月に実施した[5]。プレゼンテーションは、受け手に向けての直接的で即時的な情報発信であり、したがって、調査結果は、情報を発信することに関する全盲者など重度視覚障害者の実情を示すものである。その中からは、「図形や画像など、文字以外の視覚的表現を自身で用いる術が現在はないが、より自立的で確かな情報発信を行うためにそれらの利用に主体的に関われるようになりたい」とのニーズがあることが読み取れる。描画や作画の方法を開発する必要性の裏づけとして、同調査結果の要点を紹介する。

調査対象は、職業経験（もしくはそれに準ずる社会活動の経験）を有する（成人の）重度視覚障害者である。

回答を寄せた69人の障害程度は、64人が1級、5人が2級である。回答者の文字処理の方法を表1、職種を表2にしめす。

表1　文字処理の方法（複数回答可）

| 文字処理の方法 | 選択者数 |
|---|---|
| 点字を使用している | 64 |
| 補助具を用いずに墨字を読み書きしている | 0 |
| レンズ類を用いて墨字を読み書きしている | 2 |
| 拡大読書器を用いて墨字を読み書きしている | 4 |
| パソコンを用いて視覚で墨字を読み書きしている | 3 |
| パソコンを用いて視覚以外で墨字を読み書きしている | 64 |
| 墨字の読み書きに関して人の助けを借りることがある | 62 |
| 墨字を自分で読み書きすることは全くない | 11 |
| 文字の読み書きとともに、録音を活用している | 38 |
| 点字も墨字も読み書きせず、録音を活用している | 0 |
| その他 | 7 |

回答者の85.5%（59人）が、学校での授業などを含め、何らかのプレゼンテーションを行っている。その方法は表3の通り（複数選択）であり、口述以外での情報発信に様々にとりくんでいる。そして、提示や配布をする資料の作成では、表4のとおり、補助者への依存の度合いがいまなお高い。

一方、プレゼンテーションを行うことに関連する問題点を54人が具体的に回答した。その内容を整理・集約した結果を表5に示す。

また、プレゼンテーションを行うことを支援する機能として、表6のようなものの実現が望まれている。

表2　職種別人数

| 職種 | 人数 |
|---|---|
| 教員 | 30 |
| 指導員 | 2 |
| あんま・マッサージ・指圧師・鍼師・灸師 | 2 |
| 事務的職種 | 17 |
| 技術系職種 | 12 |
| 福祉系職種 | 2 |
| その他 | 4 |
| 計 | 69 |

表3　プレゼンテーションの方法（複数回答可）

| 方法 | 選択者数 |
|---|---|
| 口述のみで行う | 46 |
| 独力で板書を行う | 13 |
| パソコンを独力で操作し、文字などをその場で入力して提示する | 18 |
| 補助者に板書を依頼する | 19 |
| 事前に作成した紙ベースの資料を提示する | 29 |
| 事前に作成したスライドやOHPシートを独力で提示する | 3 |
| 事前に作成したスライドやOHPシートを補助者による操作で提示する | 16 |
| 事前に作成した印刷物の資料を配布する | 53 |
| 事前に作成した電子資料を独力でパソコンを操作して提示する | 21 |
| 事前に作成した電子資料を補助者によるパソコン操作で提示する | 10 |
| その他 | 10 |

この調査結果から、次のようなツールの開発が必要であると考えられる。いずれも、点字・点図(触図)・音声出力・ガイダンス音などを介してアクセスするものでなければならない。

a．即時提示用ツール：文字や簡単な図形をその場で書いて(描いて)示すための(黒板の代わりとなるような)ツール

b．資料提示用ツール：事前に作成した文書・図表・画像などの(スライド形式の)資料を、順次または選択的に提示するためのツール

c．資料作成用ツール：提示資料を作成(作図や描画、既存の画像データを利用した作画など)するためのツール

表4　資料作成時の依存状況（複数回答あり）

| 依存状況 | 選択者数 |
|---|---|
| 独力で行い、文や表など、簡素な形式の資料のみ準備している | 21 |
| すべて独力で多彩な形式の資料を準備している | 2 |
| 文の入力などは、自分で行うが、書式の整形や画像データの処理など視覚を必要とする作業は補助者に依頼している | 39 |
| 内容は決めるが、自分では作成作業は行わない | 4 |
| その他 | 4 |

表5　プレゼンテーションを行うことについての問題点

| |
|---|
| a．視覚情報の提示についての不便や困難 |
| その場で文字や図形を記して示せないのが不便 |
| 文字だけの提示では不十分 |
| 提示している資料の確認・指し示し・修正ができないのが不便 |
| 資料を配布すると聞き手がそれに注目し話を聞かなくなるのが困る |
| b．口述のみで行うことの問題点 |
| 口述だけでは伝えにくい内容がある |
| 口述だけでは聞き手の集中が持続しない |
| c．内容の組み立てや進行に関する問題点 |
| (晴眼者に対する)良いプレゼンテーションの方法がわからない |
| 点字のメモを見ながらでは機器を操作しにくい |
| 機器の操作が難しくスムーズに行えない |
| 臨機応変の時間調整が難しい |
| 補助者の確保が容易でない |
| d．資料の準備に関する問題点 |
| 視覚資料に関する自身の知識が不足している |
| 良い視覚資料の作り方がわからない |
| 電子資料の作成が難しい |
| 画像や映像の理解や把握が難しい |
| 資料作りのツールがない |

## 4．Java 言語を活用した描画の試み

Java 言語を使って非視覚的に描画を行う方法を試みた。

### 4．1　取り組みの概要

前述のように、正確・精密な描画を触覚だけに頼って行うことは困難である。そこで、二次元座標を使って画像を記述する方法を用いることとし、Java 言語の描画機能をその手段として選んだ。すなわち、「目的の画像を生成し画面上に表示するアプレットプログラムを、非視覚的な環境で作成できるようにすること」を当座の目標とした。そこで、Java の開発環境に点字出力と音声出力を導入するとともに、画像を確認する手段として点図ディスプレイと立体コピーシステムを利用することとし、可能性や問題点を明らかにするために、全盲である筆者が実際にこの方法での描画を試みた。

### 4．2　Java を用いる理由

Java は、C や C++などの利点を備え、同時にそれらの難点のいくつかが解消された、使いやすい言語である。実際 Java では、従来の言語でのプログラミング経験が活かせる上、記憶領域の管理に関する煩雑さなどから開放される。また、インターネットへの対応が容易なことも大きな魅力であり、WWW の下での様々な処理を実現できる。と

表6　必要と思われる支援機能（複数回答可）

| 機能 | 選択者数 |
|---|---|
| 文字を入力して即座に提示できる機能 | 51 |
| 数式や図形などを入力して即座に提示できる機能 | 32 |
| 提示中の文や表にアクセスできる機能 | 53 |
| 提示中の数式や図形にアクセスできる機能 | 30 |
| 事前に用意した画像データを自在に提示できる機能 | 44 |
| 事前に用意した映像資料を自在に提示できる機能 | 44 |
| その他 | 10 |

りわけ、アプレット作成機能は有用である。

そして、視覚障害者にとっての最大の利点は、従来型のコマンドライン方式でプログラム開発を行えることである[6]。Java のプログラミングには、JavaDeveloper's Kit (JDK) という開発環境を利用するが、JDK は Java の開発元の SunMicrosystems 社が無償で配布している。コンパイラやインタプリタの起動をコマンドラインのキー入力で行うのをはじめ、JDK のリソースはコマンドライン環境でアクセスできる。現在の標準的な基本ソフトである Windows の下では開発環境がほとんどすべてGUI化しているなか、JDK の、一般的には一時代前のこの方式は、視覚障害者にとって極めて貴重である。言語としての優秀性とともに、この開発環境をもつことが、Java を利用手段に選定した理由である。

### 4．3　プログラミングの方法

非視覚的な方法でのJavaプログラミングについて概説する。

#### 4．3．1　支援ソフトと機器

WindowsXP 上の JDK1.3 で全盲者がアプレットプログラミングを行えるようにするために、以下の支援ソフトと機器を用意した。

- 「コマンドプロンプト」にも対応しているスクリーンリーダ
- 点字表示機能を備えたテキストエディタ
- 点字ディスプレイ端末（点字表示部 46 セル）
- 点図ディスプレイ端末（縦 24 点・横 32 点・点間ピッチ 3mm）及び同端末用ソフト
- 立体コピーシステム

#### 4．3．2　コーディング

ソースコードの入力と編集・修正には、そのために開発した、点字表示機能付きテキストエディタ[7]を用いる。このエディタには、テキストデータを情報処理用点字で実時間表示する機能がある。また、既存のスクリーンリーダの音声読み上げ機能との併用が可能である。音声による読み上げと情報処理用点字によるディスプレイ表示を併用することにより、ソースコードの迅速で正確な入力や編集・修正が可能になる。

#### 4．3．3　コンパイル

アプレットのソースコードは、独立型のアプリケーションプログラムと同様、コンパイラ javac でコンパイルする。javac は、コマンドプロンプト上でのキー操作で起動する。この操作は、スクリーンリーダの音声読み上げの下で行える。

コンパイラからのエラーメッセージは、音声で読み上げられる。出力されたメッセージを読み返すには、スクリーンリーダのレビュー機能を利用する。しかし、エラーの件数が多い場合など、大量の出力に対しては、音声読み上げでは対応しにくい。また、出力内容によっては、音声だけでは細部まで正確に理解できないことがある。そのような場合は、メッセージ出力を点字に変換して読む。すなわち、エラーメッセージをリダイレクト出力でファイルに書き出して保存し、それをテキストエディタに読み込んで、情報処理用点字による表示機能を使って点字ディスプレイ端末上で触読する。ただし、javac からのエラーメッセージは、通常、標準エラー出力に書き出されるので、そのままではリダイレクトでファイルに書き出すことができない。そこで、ファイルへの書き出しを行うために、javac の起動時に、出力チャネルの切り替えを指示する必要がある。さらに、JDK1.3 ではその切り替えを行う機能がないため、旧版の javac を呼び出す機能を利用し、その上で出力チャネルの切り替えをしなければならない。

#### 4．3．4　アプレットの動作確認とプログラムの修正

アプレットの動作確認には、そのためのツールである appletviewer を用いる。それに先立って、テキストエディタで当該アプレット用の HTML ファイルを作成し、そのファイル名を引数にして、コマンドプロンプト上で appletviewer を起動する。当然、この起動操作はスクリーンリーダの読み上げの下で行える。

アプレットの画像表示の確認には、点図ディスプレイ端末が使用できる。これにより、実時間での確認が可能である。しかし、点図表示部の広さや点の密度が十分でないことから、細部の確認はできない。そこで、表示画像を印刷し、立体コピーシステムで触図にして確認する方法を併用する。多少の手間と時間がかかるが、これによってかなり確かな確認ができるので、それを参照しながら、ソースコードを修正する。

### 4．4　Javaの描画機能

Java言語には、二次元座標で位置や大きさを指定して線分、多角形、円（楕円）などの基本図形を描く一群のメソッドが用意されており、基本ライブラリ中のクラスに集約されている。彩色などもメソッドの引数で自在に指定でき、全盲者にも十分に利用可能である。アプレットのソースコード中でこれらのメソッドを使い、画像を記述する。表7にソースコードの例を示す。

### 4．5　画像の保存と表示

作成したアプレットをバイトコード形式で保存しておけば、インターネットブラウザなどでいつでもすぐにその画像を表示できる。また、画像をビットマップデータとして保存すれば、プレゼンテーション用ソフトなど、他のソフトでも利用できる。

### 4．6　描画の例

図1は、筆者がこの方法で最初に描いた画像を線画化したものである。実際には、まず、この線画を作成し、それに彩色をして最終的な画像にした。線画の作成過程では、立体コピーシステムでの触図を参照して3回の調整（ソースコードの修正による配置や大きさの変更）を行った。表7はこの画像を記述したソースコードの一部である。

表7　図1の原画像を描くJavaソースコードの一部

```
// 屋根の位置と外形の設定
int yaneX[] = {150, 350, 400, 100};
  int yaneY[] = {100, 100, 175, 175};
// 旗の位置と外形の設定
  int hataX[] = {85, 150, 150};
  int hataY[] = {45, 30, 60};
// 背景色の設定
    setBackground(Color.white);
// 屋根の描画
    g.setColor(Color.red);
    g.fillPolygon(yaneX, yaneY, 4);
    g.setColor(Color.black);
    for (int i = 1; i < 10; i++) {
      g.drawLine(150+20*i, 100, 100+30*i, 175);
    }
// 煙突の描画
    g.fillRect(200, 60, 30, 40);
// 旗の描画
    g.drawLine(150, 100, 150, 30);
    g.setColor(Color.cyan);
    g.fillPolygon(hataX, hataY, 3);
// 月の描画
    g.setColor(Color.pink);
    g.fillArc(410, 20, 60, 60, -90, 180);
    g.setColor(Color.white);
    g.fillArc(425, 20, 30, 60, -90, 180);
```

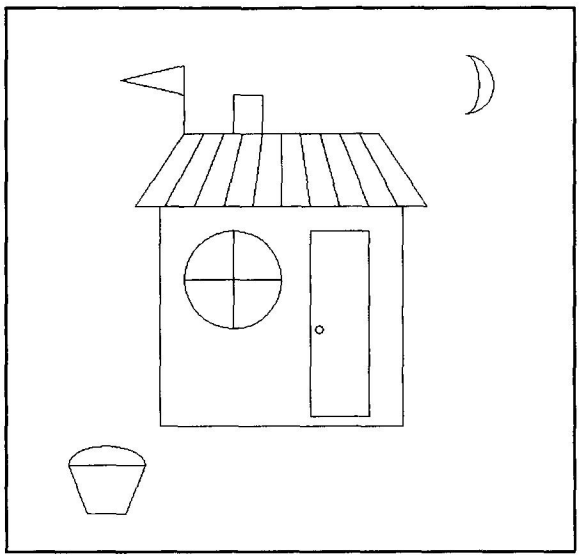


図1　視覚を用いずにJavaで描いた画像の輪郭

## 5．問題点と課題

視覚を用いずに描画を行うこの方法には、改善が必要な問題点がいくつかある。その主なものと、それらの改善を図るための方向性について記す。

### 5．1　文法誤り訂正過程の非効率性

コンパイラからのエラーメッセージに基づくソースコードの訂正を、音声読み上げだけに頼って行うことには、能率や精度の点からかなりの無理がある。そこで、エラーメッセージの点字変換が必要になる。しかし、エラーメッセージを点字で読むための現在の方法は、決して望ましいものではない。すなわち、旧版のコンパイラを使用しなければならないうえ、エラーメッセージとソースコードが別のファイルになり、作業の効率が悪い。

その改善には、以下のようなことが必要である。

- エラーメッセージの合理的な取得方法の確立
- コンパイラとエディタの連動化
- 点字表示機能の向上

### 5．2　画像の確認・修正過程の非効率性

作成した画像の確認は、アプレットを実行し、点図ディスプレイ端末もしくは立体コピーシステムによって行う。このとき、前者は即時的な確認ができるが、表示の精度が悪い。後者は、比較的正確に確認ができるものの、時間と手間がかかる。いずれにしても、画像の確認と修正の作業は、効率が悪い。

前者については、現有機よりも表示部が広くピンの密度が高い高機能機を導入することによってある程度の改善が図れると期待できる。後者については、現在のところ改善の余地はない。最終的には、次の二つのものの開発が望まれるところである。

- PC のモニタに近い精細度で表示できる触覚ディスプレイ端末
- ソースコードの修正が表示中の画像（モニタと触覚ディスプレイ）に即座に反映され、それを実時間で確認できるヒューマンインタフェース機能

### 5．3　描画の限界

線分、多角形、円などの基本図形を組み合わせることによる描画は、この方法でかなりの程度可能である。しかし、写実的な描画など、基本図形では表しえない対象を描くことは難しい。

このことへの対応策の一つとして、画像のライブラリ化が考えられる。すなわち、各種の画像を蓄積しておき、それらの既存画像を利用できるようにする。「既存の画像に手を加える」、「自身で描く画像の中に既存の画像を取り入れる」、「既存の画像を組み合わせて新たな画像を作る」など、「描画」というよりも「作画」を可能にする方法である。また、分野別の画像ライブラリがあっても良い。

一方、(Java プログラムのように) 言語で画像を記述する方式と、渡辺・小林による「電子レーズライタ」のように直接に画像を描く方式の併用も、検討に値する。これらのいずれにおいても、画像の修正を即座に確認できる機能が必要である[8]。

### 5．4　記述の難解さ

Java の既存の描画メソッドをそのまま使って画像を記述するのは容易でない。「基本図形用のメソッドしかないこと」、「引数が煩雑なこと」、「メソッドの名称が一般にはなじみにくいこと」、などがその理由である。

その改善には、画像を簡便に記述できる描画用クラスやメソッド、さらには言語の開発が必要である。分野別専用の描画クラスやメソッドの開発も、検討に値する。前述の分野別画像ライブラリと組み合わせることにより、各自の専門分野に関連する作画なら、ある程度独力で行えるようになる可能性も期待できる。

## 6．おわりに

基本図形を組み合わせる簡単な描画ならば、Java 言語を用いて非視覚的な方法で行うことができた。しかし、その可能性を拡大し、一般性のある方法として確立するには、多くの解決すべき課題がある。

課題は、技術的な開発だけではない。どのような方法にせよ全盲者が描画や作画を行えるようになるには、図形や画像に関する知識や経験を十分に持つことが重要である。その中には、色彩についての基本的な知識なども含まれる。それを実現するためには、教育の果たす役割が大きい。

情報化社会の中で、全盲者も情報の発信者となって積極的に社会参加できることが望ましい。そのためには、多くの人に受け入れられる方法で情報を発信できる技能を身につける必要がある。今後、それに関連する技術開発や教育の充実が進むことを願い、その一助となるよう、本取り組みを継続・発展させたい。

＜参考文献＞


[1] 渡辺哲也：視覚障害者の Windows パソコン及びインターネット利用・学習状況, 独立行政法人国立特殊教育総合研究所報告書 D-190, 2003.

[2] 原田政美・小柳恭治：盲人用レーズライターの試作とその性能に関する実験, 東北大学教育学部 研究年報 XV, 225-243, 1967.

[3] 渡辺哲也・小林真：盲学校における電子レーズライタ MIMIZU の評価, 電子情報通信学会技術報告, WIT2003-2, 2003.

[4] 藤芳　衛：視覚障害者の触読図の作成を可能にするプロッタ・システムの開発, 電子情報通信学会技術研究報告, Vol.103, No.746, 7-12, 2004.

[5] 長岡英司　他：重度視覚障害者によるプレゼンテーションの現状,電子情報通信学会技術研究報告, Vol.204, No.637,TL2004-40, 2005.

[6] 長岡英司：重度視覚障害者による Java プログラミングの可能性、筑波技術短期大学テクノレポート, Vol.12,21-26, 2005.

[7] 長岡英司・星加恒夫：実時間点字表示機能を備えたテキストディタの開発と活用,第 30 回感覚代行シンポジウム講演論文集,99-104, 2004.

[8] 清水豊・篠原正美　他：Improvement of User Interface for Blind PC Users, ICCHP, Proceedings of 8th International Conference on Computers Helping People with Special Needs,2002

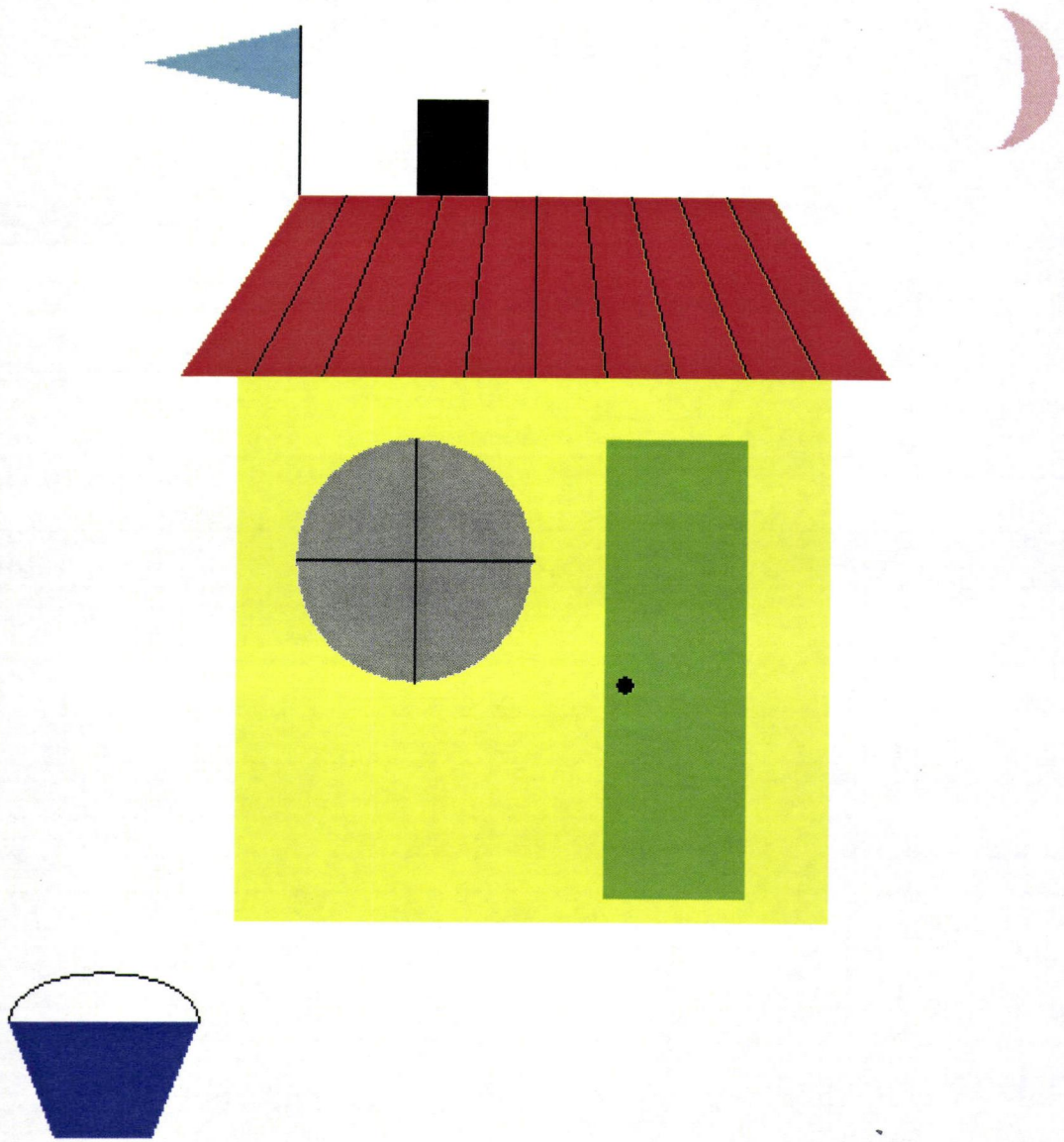


＊ この取り組みは、文部科学省科学研究費補助金（基盤研究（C）17500670）による研究の一環として行われた。

# プログラミング環境

# レーズライター

# 図1の原図



# 付　録

## AiB　Tools　ソースコードの抜粋

AiB Tools は 2006 年 3 月 17 日の時点でソースコードが 27195 行に達しているため、AiB Edit のソースコードから主要なものだけを取り出してここに掲載する。掲載したソースコードはファイルごとに次のような役割の実装部分となっている。

- AppLogic.cs: テキストエディタ AiB Edit の主要なロジックを実装
- AiBEditForm.cs: テキストエディタ AiB Edit のメインウィンドウを実装
- CppOutliner.cs: C/C++言語のソースコードを解析してアウトラインを抽出するアルゴリズム
- RubyOutliner.cs: Ruby 言語のソースコードを解析してアウトラインを抽出するアルゴリズム
- AutoSpeaker.cs: 各種日本語スクリーンリーダーの外部制御 API を C#から使うためのライブラリ

## AppLogic.cs

```
// file     : AppLogic.cs
// brief    : application logic
// update   : 2007-03-10
//==========================================================
using System;
using System.Collections;
using System.Text;
using System.Text.RegularExpressions;
using System.IO;
using System.Windows.Forms;
using CancelEventArgs = System.ComponentModel.CancelEventArgs;
using CancelEventHandler = System.ComponentModel.CancelEventHandler;
using Sgry.Nbs;
using Sgry.Nbs.Forms;

namespace Sgry.AiBTools.AiBEdit
{
    /// <summary>
    /// AiB Edit のアプリケーションロジック。
    /// MVC モデルでの Controller の役割を担い、
    /// View である AiBEditForm と Model である Document の間で処理の受け渡しをする。
    /// </summary>
    /// <remarks>
    /// インスタンス生成後に View (AiBEditForm) を指定しないと正常に動作しない
    /// </remarks>
    internal class AppLogic
    {
        #region Fields
        const string FileSelectionFilter = "All files(*.*)|*.*|Text files(*.java,
*.c,
...)|*.txt;*.tex;*.java;*.rb;*.pl;*.py;*.c;*.cpp;*.cxx;*.cs;*.h;*.hpp;*.hxx;*.vbs;*.b
at;*.ini;*.inf";
        AiBEditForm         _View;
        ArrayList           _Documents      = new ArrayList( 10 );
        Document            _ActiveDocument;
        Localizer           _Localizer      = new Localizer();
        DeviceConfig        _DeviceConfig   = new DeviceConfig();
```

```
        BrailleConfig        _BrailleConfig  = new BrailleConfig();
        bool                 _ProcessingActivatedEvent   = false;
        RichEdit20WBrailler _TextBoxBrailler;
        int                  _UntitledDocumentNum = 1;
int                 _SwitchingDocument_Index = 0;
        FindCondition        _FindCondition  = new FindCondition();
        Hashtable            _DocMenuMap     = new Hashtable();
        #endregion

        #region Init / Dispose
        /// <summary>
        /// 新しいインスタンスを生成
        /// </summary>
        public AppLogic()
        {
            AppConfig.Init();
            _Localizer.LoadResourceFile();
        }

        /// <summary>
        /// メインウィンドウを設定
        /// </summary>
        /// <param name="form">メインウィンドウ</param>
        public void SetMainWindow( AiBEditForm form )
        {
            _View = form;
            _View.Closing += new CancelEventHandler( Form_Closing );
            _View.Closed += new EventHandler( Form_Closed );
            _View.Activated += new EventHandler( Form_Activated );
            _View.Deactivate += new EventHandler( Form_Deactivate );

            _View.TabControl.TabClicked += new AiBTabControl.TabClickedEventHandler(
TabControl_TabClicked );

            // ファイル種別をメニューに追加
            foreach( IFileType fileType in FileTypeRepository.AllFileTypes )
            {
                MenuItem mi = new MenuItem();
                mi.Text = fileType.Name;
                mi.Tag = fileType;
                mi.Click += new EventHandler( EditModeMenuItem_Click );
                _View.EditModeMenu.MenuItems.Add( mi );
            }

            // 「最近使ったファイル」メニューを更新
            UpdateMruFileMenu();
        }
        #endregion

        #region Menu/Dialog control
```

```
/// <summary>
/// 「最近使ったファイル」メニューを更新
/// </summary>
void UpdateMruFileMenu()
{
    string[]    mru;
    string      fileName;
    MenuItem    mi;

    // MRU 情報を取得
    mru = AppConfig.Get17MruFilePaths();
    if( mru.Length == 0 )
    {
        _View.MruMenu.Enabled = false;
        return;
    }

    // メニューを更新
    _View.MruMenu.Enabled = true;
    _View.MruMenu.MenuItems.Clear();
    for( int mruIndex=0; mruIndex<mru.Length; mruIndex++ )
    {
        string mruFilePath = mru[mruIndex];
        fileName = Path.GetFileName( mruFilePath );
        mi = new MenuItem();
        mi.Text = String.Format( "{0} {1}", mruIndex.ToString("X"), fileName );
        mi.Tag = mruFilePath;
        mi.Click += new EventHandler( MruFileMenu_Clicked );
        _View.MruMenu.MenuItems.Add( mi );
    }
}

/// <summary>
/// MRU ファイルのメニュー項目が選択された時の動作。
/// </summary>
void MruFileMenu_Clicked( object sender, EventArgs e )
{
    MenuItem mi = (MenuItem)sender;
    string filePath = (string)mi.Tag;

    OpenDocument( filePath );
}

/// <summary>
/// 「編集モード」メニューの表示を更新
/// </summary>
void UpdateEditModeMenu()
{
    foreach( MenuItem mi in _View.EditModeMenu.MenuItems )
    {
```

```
            // 各ファイル種オブジェクトは唯一なのでハンドル値で同定
            if( ActiveDocument.FileType.GetHashCode() == mi.Tag.GetHashCode() )
                mi.Checked = true;
            else
                mi.Checked = false;
        }
    }

    /// <summary>
    /// アウトラインダイアログを表示
    /// </summary>
    public void ShowOutlineDialog()
    {
        int newPos;

        // show dialog
        newPos = ActiveDocument.FileType.ShowOutline( _View,
ActiveDocument.TextBox.Text, ActiveDocument.TextBox.GetCaretIndex() );

        // reset braille config anyway
        NbsEngine.Instance.BrailleConfig = _BrailleConfig;

        // jump to the selected symbol
        if( 0 < newPos )
            ActiveDocument.SetCaretIndex(
ActiveDocument.TextBox.GetLineFromCharIndex(newPos), 0 );
    }
    #endregion

    #region MDI 処理
    /// <summary>
    /// 個々の書類用メニュー項目がクリックされた直後の動作
    /// </summary>
    void DocMenu_Click( object sender, EventArgs e )
    {
        MenuItem mi = (MenuItem)sender;
        SetActiveDocument( (Document)mi.Tag );
    }

    /// <summary>
    /// 書類のタブがクリックされた直後の動作。
    /// タブに関連付けられた書類を
    /// 左クリックならアクティブに、
    /// 中クリックなら閉じる。
    /// </summary>
    void TabControl_TabClicked( MouseButtons button, int index, object tag )
    {
        if( button == MouseButtons.Left )
        {
            SetActiveDocument( (Document)tag );
```

```
    }
    else if( button == MouseButtons.Middle )
    {
        Win32.LockWindowUpdate( _View.Handle );
        SetActiveDocument( (Document)tag );
        Win32.LockWindowUpdate( IntPtr.Zero );
        CloseDocument();
    }
}

/// <summary>
/// アクティブな書類を取得
/// </summary>
public Document ActiveDocument
{
    get{ return _ActiveDocument; }
}

/// <summary>
/// 書類の切り替えを開始
/// </summary>
/// <remarks>実際には何もしない</remarks>
public void BeginSwitchingDocument()
{}

/// <summary>
/// 書類の切り替えを終了。
/// 書類を並び替える。
/// </summary>
public void EndSwitchingDocument()
{
    // 最終的にアクティブになった書類をリストの先頭にする
    Document activatedDoc = (Document)_Documents[_SwitchingDocument_Index];
    _Documents.RemoveAt( _SwitchingDocument_Index );
    _Documents.Insert( 0, activatedDoc );

    _SwitchingDocument_Index = 0;
}

/// <summary>
/// 次の書類に切り替える。
/// </summary>
public void SwitchToNextDocument()
{
    if( _SwitchingDocument_Index < _Documents.Count )
        _SwitchingDocument_Index++;
    else
        _SwitchingDocument_Index = 0;
    SetActiveDocument( (Document)_Documents[_SwitchingDocument_Index] );
}
```

```
/// <summary>
/// 前の書類に切り替える。
/// </summary>
public void SwitchToPreviousDocument()
{
    if( 0 < _SwitchingDocument_Index )
        _SwitchingDocument_Index--;
    else
        _SwitchingDocument_Index = _Documents.Count - 1;
    SetActiveDocument( (Document)_Documents[_SwitchingDocument_Index] );
}

/// <summary>
/// 書類の一覧を表示
/// </summary>
public void ShowDocumentList()
{
    DialogResult        result;
    DocumentListDialog  dialog;

    if( _Documents.Count == 0 )
    {
        Win32.MessageBeep_Notify();
        return;
    }

    // show dialog
    dialog = new DocumentListDialog( _Documents, ActiveDocument );
    result = dialog.ShowDialog( _View );
    if( result == DialogResult.OK )
    {
        SetActiveDocument( dialog.SelectedDocument );
    }
}

/// <summary>
/// エディタ上でアクティブな書類を変更
/// </summary>
public void SetActiveDocument( int docHashCode )
{
    foreach( Document doc in _Documents )
    {
        if( doc.GetHashCode() == docHashCode )
        {
            SetActiveDocument( doc );
            return;
        }
    }
}
```

```
/// <summary>
/// エディタ上でアクティブな書類を変更
/// </summary>
public void SetActiveDocument( Document doc )
{
    // アクティブでなくなる書類用のウィンドウを隠す
    if( _ActiveDocument != null )
    {
        _ActiveDocument.TextBox.Hide();
        ((MenuItem)_DocMenuMap[_ActiveDocument]).Checked = false;
    }

    _ActiveDocument = doc;

    // アクティブになった書類のウィンドウを表示
    ActiveDocument.TextBox.Show();
    ActiveDocument.TextBox.Select();

    // タブを選択
    if( _View.TabControl.Contains(ActiveDocument) )
    {
        _View.TabControl.Select( ActiveDocument );
        _View.TabControl.Refresh();
    }

    // 点字ディスプレイでこの書類を表示するように
    if( _TextBoxBrailler != null )
    {
        _TextBoxBrailler.ActiveControl = (RichEdit20W)ActiveDocument.TextBox;
        _TextBoxBrailler.UpdateDisplay();
    }

    // メインフォームの UI を更新
    _View.ResetText();
    ((MenuItem)_DocMenuMap[ActiveDocument]).Checked = true;
    _View.EnableSaveReloadMenuItem( (ActiveDocument.Path != null) );
    UpdateEditModeMenu();
}
#endregion

#region 書類の処理
/// <summary>
/// 新しく書類を生成
/// </summary>
public void CreateDocument()
{
    Document doc = new Document();
    MenuItem mi = new MenuItem();
```

```
        // setup text box
        _View.TextBoxPanel.Controls.Add( doc.TextBox );
        doc.TextBox.Dock = DockStyle.Fill;
        doc.ContentModified += new EventHandler( Document_ContentModified );

        // setup window menu
        mi.Tag = doc;
        mi.Click += new EventHandler( DocMenu_Click );
        _View.WindowMenu.MenuItems.Add( mi );
        _DocMenuMap.Add( doc, mi );

        // set default file type
        doc.FileType = FileTypeRepository.GetByTypeName( null );

        // add document list
        _Documents.Insert( 0, doc );

        // activate
        SetActiveDocument( doc );
    }

    /// <summary>
    /// 編集モードのメニュー項目がクリックされた直後の動作
    /// </summary>
    void EditModeMenuItem_Click( object sender, EventArgs e )
    {
        MenuItem mi = (MenuItem)sender;
        ActiveDocument.FileType = (IFileType)mi.Tag;

        // チェックを付け直す
        foreach( MenuItem i in _View.EditModeMenu.MenuItems )
        {
            i.Checked = false;
        }
        mi.Checked = true;
    }

    /// <summary>
    /// 新しく無名の書類を生成
    /// </summary>
    public void CreateUntitledDocument()
    {
        CreateDocument();
        ActiveDocument.Title = "Untitled" + _UntitledDocumentNum++;

        // UI を更新
        ((MenuItem)_DocMenuMap[ActiveDocument]).Text
            = String.Format( "&{0} {1}", _View.WindowMenu.MenuItems.Count-4,
ActiveDocument.Title );
        _View.ResetText();
```

```
    _View.TabControl.Add( ActiveDocument );
    _View.TabControl.Select( ActiveDocument );
    if( _Documents.Count == 1 )
    {
        _View.EnableEditMenuItems( true, true );
        _View.EnableWindowSwitchMenuItems( true );
    }
    _View.EnableSaveReloadMenuItem( (ActiveDocument.Path != null) );
}

/// <summary>
/// 書類を閉じる
/// </summary>
/// <returns>
/// 変更済み書類を閉じるのをユーザがキャンセルしたら true
/// </returns>
public bool CloseDocument()
{
    Document doc = ActiveDocument;

    // if no documents exists, do nothing
    if( _Documents.Count == 0 )
    {
        return false;
    }

    // if already modified, ask user to save it or not.
    if( doc.Modified )
    {
        DialogResult result = AskToSaveDocument();
        if( result == DialogResult.Yes )
        {
            result = SaveDocument();
            if( result == DialogResult.Cancel )
            {
                return true;
            }
        }
        else if( result == DialogResult.Cancel )
        {
            return true;
        }
    }

    // remove UI
    doc.TextBox.Enabled = false;
    _ActiveDocument = null;
    if( _TextBoxBrailler != null )
        _TextBoxBrailler.ActiveControl = null;
    _View.WindowMenu.MenuItems.Remove( (MenuItem)_DocMenuMap[doc] );
```

```
        _View.TabControl.Remove( doc );
        _View.Controls.Remove( doc.TextBox );

        // remove from document list
        _Documents.Remove( doc );

        // activate other
        if( 0 < _Documents.Count )
        {
            SetActiveDocument( (Document)_Documents[0] );
        }
        else
        {
            _View.ResetText();
            _View.EnableEditMenuItems( false, true );
            _View.EnableWindowSwitchMenuItems( false );
            if( NbsEngine.Instance != null )
                NbsEngine.Instance.ClearDisplay();
        }

        // dispose document resource
        doc.Dispose();

        return false;
    }

    /// <summary>
    /// ファイルを開く
    /// </summary>
    public void OpenDocument()
    {
        DialogResult    result;
        OpenFileDialog  dialog;

        using( dialog = new OpenFileDialog() )
        {
            // setup dialog window
            dialog.StartPosition = FormStartPosition.CenterParent;
            dialog.Filter = FileSelectionFilter;
            dialog.FilterIndex = 1;
            if( ActiveDocument != null )
            {
                // set initial value from the active document
                if( File.Exists(ActiveDocument.Path) )
                {
                    dialog.InitialDirectory = Path.GetDirectoryName(
ActiveDocument.Path );
                }
                dialog.FileName = ActiveDocument.FileName;
            }
```

```
        // show dialog
        result = dialog.ShowDialog( _View );
        if( result == DialogResult.Cancel )
        {
            return;
        }

        // if specified file doesn't exist, create empty file.
        if( File.Exists(dialog.FileName) == false )
        {
            File.Create( dialog.FileName ).Close();
        }

        // open chosen file
        if( dialog.Encoding == null )
        {
            OpenDocument( dialog.FileName );
        }
        else
        {
            OpenDocument( dialog.FileName,
                dialog.Encoding, dialog.WithBom,
                -1, -1, null );
        }

        // refresh display
        _View.ResetText();
        if( _TextBoxBrailler != null )
            _TextBoxBrailler.UpdateDisplay();
    }
}

/// <summary>
/// ファイルを開く
/// </summary>
public void OpenDocument( string filePath )
{
    OpenDocument( filePath, null, false, -1, -1, null );
}

/// <summary>
/// ファイルを開く
/// </summary>
void OpenDocument( string filePath, Encoding encoding, bool withBom,
    int caretLine, int caretColumn, IFileType fileType )
{
    // 無指定のパラメータを MRU 情報で上書き
    {
        Encoding encoding_;
```

```
        bool withBom_;
        int caretLine_, caretColumn_;

        AppConfig.GetMruFileInfo( filePath,
            out encoding_, out withBom_, out caretLine_, out caretColumn_ );
        if( encoding == null )
        {
            encoding = encoding_;
            withBom = withBom_;
        }
        if( caretLine < 0 ) caretLine = caretLine_;
        if( caretColumn < 0 ) caretColumn = caretColumn_;
    }

    filePath = ReGetFilePath( filePath );

    try
    {
        // ユーザが選択したファイルを編集中ならそれをアクティブにする
        foreach( Document doc in _Documents )
        {
            if( doc.Path == filePath )
            {
                SetActiveDocument( doc );
                return;
            }
        }

        // ちゃんと開く事ができるか確認
        File.OpenRead( filePath ).Close();

        // 未編集でなければ新しい書類を生成
        if( ActiveDocument == null
            || 0 < ActiveDocument.ContentLength
            || ActiveDocument.Modified )
        {
            CreateDocument();
        }

        // 開く
        _View.TabControl.Remove( ActiveDocument );
        ActiveDocument.Open( filePath, encoding, withBom );

        // 書類のプロパティを適用
        if( fileType == null )
        {
            fileType = FileTypeRepository.GetByFilePath( filePath );
        }
        ActiveDocument.FileType = fileType;
        ActiveDocument.SetCaretIndex( caretLine, caretColumn );
```

```
            // MRU を更新
            AppConfig.AddMru( ActiveDocument );

            // UI を更新
            ((MenuItem)_DocMenuMap[ActiveDocument]).Text
                = String.Format( "&{0} {1}", _View.WindowMenu.MenuItems.Count-4,
ActiveDocument.Title );
            _View.TabControl.Add( ActiveDocument );
            _View.TabControl.Select( ActiveDocument );
            if( _Documents.Count == 1 )
            {
                _View.EnableEditMenuItems( true, true );
                _View.EnableWindowSwitchMenuItems( true );
            }
            UpdateEditModeMenu();
            UpdateMruFileMenu();
            _View.ResetText();

            // 新しく開いた書類の名前を読み上げる
            if( AutoSpeaker.IsAlive() )
            {
                ISpeaker speaker = AutoSpeaker.Instance;
                speaker.Stop();
                speaker.Speak( ActiveDocument.Title );
            }
        }
        catch( IOException ex )
        {
            AlertFailedToOpenFile( ex );
        }
    }

    /// <summary>
    /// この窓でファイルを読み直す。
    /// エンコーディング等は再推定する。
    /// </summary>
    public void ReloadDocument()
    {
        ReloadDocument( null, false );
    }

    /// <summary>
    /// この窓でファイルを読み直す
    /// </summary>
    /// <param name="encoding">読み直すのに使うエンコーディング</param>
    public void ReloadDocument( Encoding encoding, bool withBom )
    {
        // 一度も保存されていない新規ファイルならば何もしない
        if( ActiveDocument.Path == null )
```

```
    {
        Win32.MessageBeep_Notify();
        return;
    }

    // 保存されていない場合は読み直して良いか尋ねる
    if( ActiveDocument.Modified )
    {
        DialogResult result = ConfirmDiscardChanges();
        if( result != DialogResult.Yes )
        {
            return;
        }
    }
    ActiveDocument.Reload( encoding, withBom );
    _View.ResetText();
    _View.TabControl.Refresh();
}

/// <summary>
/// ファイルを上書き保存
/// </summary>
/// <returns>表示した保存ダイアログに対するユーザの反応</returns>
public DialogResult SaveDocument()
{
    if( ActiveDocument.Path == String.Empty )
    {
        // ファイルを開いていない、または新規作成した内容を保存していない。
        // save as する
        return SaveDocumentAs();
    }
    else if( ActiveDocument.Modified )
    {
        // 内容が変更されているので上書き保存
        ActiveDocument.Save();

        // UI を更新
        ((MenuItem)_DocMenuMap[ActiveDocument]).Text = ActiveDocument.Title;
        _View.ResetText();
        _View.TabControl.Refresh();
        return DialogResult.OK;
    }

    return DialogResult.OK;
}

/// <summary>
/// ファイルを別名で保存
/// </summary>
/// <returns>表示した保存ダイアログに対するユーザの反応</returns>
```

```
        public DialogResult SaveDocumentAs()
        {
            DialogResult    result;
            SaveFileDialog  dialog;

            using( dialog = new SaveFileDialog() )
            {
                // setup dialog window
                dialog.StartPosition = FormStartPosition.CenterParent;
                dialog.FileName = ActiveDocument.FileName;
                dialog.Filter = FileSelectionFilter;
                dialog.Encoding = ActiveDocument.Encoding;
                dialog.WithBom = ActiveDocument.WithBom;
                dialog.EolCode = ActiveDocument.EolCode;
                if( File.Exists(ActiveDocument.Path) )
                {
                    dialog.InitialDirectory = Path.GetDirectoryName( ActiveDocument.Path
);
                }

                // show save dialog
                result = dialog.ShowDialog( _View );
                if( result == DialogResult.Cancel )
                {
                    return DialogResult.Cancel;
                }

                // save as specified file
                ActiveDocument.Save( dialog.FileName, dialog.Encoding, dialog.WithBom,
dialog.EolCode );

                // judge file type
                ActiveDocument.FileType = FileTypeRepository.GetByFilePath(
dialog.FileName );

                // save MRU info
                AppConfig.AddMru( ActiveDocument );

                // update UI
                ((MenuItem)_DocMenuMap[ActiveDocument]).Text = ActiveDocument.Title;
                UpdateEditModeMenu();
                UpdateMruFileMenu();
                _View.ResetText();
                _View.TabControl.Refresh();
            }

            return DialogResult.OK;
        }

        /// <summary>
```

```
        /// 書類が変更された時の動作
        /// </summary>
        void Document_ContentModified( object sender, EventArgs e )
        {
            Document doc = (Document)sender;
            ((MenuItem)_DocMenuMap[doc]).Text = doc.Title;
            _View.ResetText();
            _View.TabControl.Refresh();
        }

        /// <summary>
        /// ファイルを開けなかった事をユーザに通知
        /// </summary>
        /// <param name="ex">開けなかった原因である例外</param>
        void AlertFailedToOpenFile( Exception ex )
        {
            string format = _Localizer.TryGetString( "AiBEditForm.Msg_FailedToOpenFile",
"Failed to open file: {0}" );
            NbsMessageBox.Show( _View, String.Format(format, ex.Message),
MessageBoxButtons.OK, MessageBoxIcon.Error );
        }

        DialogResult ConfirmDiscardChanges()
        {
            string format = _Localizer.TryGetString( "AiBEditForm.Msg_DiscardChanges",
"{0} was changed. Are you sure to discard changes?" );

            return NbsMessageBox.Show( _View, String.Format(format,
ActiveDocument.FileName), MessageBoxButtons.YesNo, MessageBoxIcon.Warning );
        }

        DialogResult AskToSaveDocument()
        {
            string          message;

            // 問い合わせメッセージの文面を生成
            if( ActiveDocument.Path == null || ActiveDocument.Path == String.Empty )
            {
                message = _Localizer.TryGetString( "AiBEditForm.Msg_NotSavedYetNew",
                        "This file has not been saved. Save before close?"
                    );
            }
            else
            {
                string format = _Localizer.TryGetString( "AiBEditForm.Msg_NotSavedYet",
                        "{0} has been modified. Save before close?"
                    );
                message = String.Format( format, Path.GetFileName(ActiveDocument.Path) );
            }
```

```
            // 問い合わせ
            return NbsMessageBox.Show( _View, message, MessageBoxButtons.YesNoCancel,
MessageBoxIcon.Information );
        }
        #endregion

        #region メインフォームのイベント処理
        /// <summary>
        /// メインフォームが閉じられる直前の確認動作。
        /// 本当に終了していいのかユーザに問い合わせる。
        /// </summary>
        void Form_Closing( object sender, CancelEventArgs e )
        {
            // すべての書類について未保存であれば保存しないのか問い合わせる
            int docCount = _Documents.Count;
            for( int i=0; i<docCount; i++ )
            {
                bool    cancel;
                cancel = CloseDocument();
                if( cancel )
                {
                    e.Cancel = true;
                    return;
                }
            }
        }

        /// <summary>
        /// メインフォームが閉じた直後の動作。
        /// 各種終了処理を実行。
        /// </summary>
        void Form_Closed( object sender, EventArgs e )
        {
            // close document
            if( ActiveDocument != null )
                ActiveDocument.Dispose();

            // save settings
            AppConfig.SaveConfig( _View, _BrailleConfig, _DeviceConfig, _FindCondition
);

            // dispose staffs around braille display
            EndUsingBraille();
            NbsEngine.Dispose();
        }

        /// <summary>
        /// フォームがアクティブになった直後の動作。
        /// 非アクティブだった間に他アプリが編集中ファイルを書き換えていたら、再読込するかユ
ーザに確認する。
```

```
/// また、アクティブでなくなった時に解放した点字ディスプレイ使用権を再取得する。
/// </summary>
void Form_Activated( object sender, EventArgs e )
{
    //---- 他アプリのファイル書き換え後時に再読込するか確認する----
    // 再読込の確認メッセージボックスを表示して閉じると、フォームがまたアクティブになる。
    // つまりイベントを処理するや否や再発生して無限ループ的な挙動になる。
    // それを回避するため、フラグを立てて複数回ここに入らないようにする
    if( _ProcessingActivatedEvent != true )
    {
        _ProcessingActivatedEvent = true;

        // アクティブでなくなる時に保存した、
        // 開いているファイルを最後に保存した時刻と、
        // 実際の最終書き込み時刻が異なれば他アプリの書き換えがあったと判断
        foreach( Document doc in _Documents )
        {
            if( File.Exists(doc.Path)
                && doc.LastSaveTime < File.GetLastWriteTime(doc.Path) )
            {
                DialogResult result = Form_Activated_ConfirmReload( doc );
                if( result == DialogResult.Yes )
                {
                    int lineIndex, columnIndex;
                    SetActiveDocument( doc );
                    Win32.LockWindowUpdate( doc.TextBox.TextBoxHandle );
                    doc.GetCaretIndex( out lineIndex, out columnIndex );
                    ReloadDocument();
                    doc.SetCaretIndex( lineIndex, columnIndex );
                    Win32.LockWindowUpdate( IntPtr.Zero );
                }
            }
        }

        _ProcessingActivatedEvent = false;
    }

    //---- 点字ディスプレイの制御を再開----
    if( _View.UsingBraille
        && NbsEngine.Instance != null )
    {
        try
        {
            if( NbsEngine.Instance.IsAttaching == false )
            {
                NbsEngine.Instance.AttachToServer( _DeviceConfig );
            }
            NbsEngine.Instance.Lock();
            if( _TextBoxBrailler != null )
```

```
                {
                    _TextBoxBrailler.UpdateDisplay();
                }
            }
            catch( NbsException )
            {}
        }
    }

    /// <summary>
    /// ユーザにファイルを読み直すかどうか確認
    /// </summary>
    DialogResult Form_Activated_ConfirmReload( Document doc )
    {
        string  format;
        string  message;

        // メッセージを生成
        format = _Localizer.TryGetString(
                "AiBEditForm.Msg_UpdatedByOtherProcess",
                "{0} was modified by other program. Reload file?"
            );
        message = String.Format( format, Path.GetFileName(doc.Path) );

        // ファイルを読み直すかどうかユーザに問い合わせる
        return NbsMessageBox.Show( _View, message, MessageBoxButtons.YesNo,
MessageBoxIcon.Warning );
    }

    /// <summary>
    /// フォームが非アクティブになる直前の動作。
    /// 他アプリのために点字ディスプレイ使用権を解放する。
    /// </summary>
    void Form_Deactivate( object sender, EventArgs e )
    {
        // 点字ディスプレイの使用権を解放
        if( _View.UsingBraille
            && NbsEngine.Instance != null
            && NbsEngine.Instance.IsAttaching )
        {
            NbsEngine.Instance.Unlock();
        }
    }
    #endregion

    #region 設定の保存と読み出し
    public void LoadConfig()
    {
        AppConfig.LoadConfig( _View, ref _BrailleConfig, ref _DeviceConfig, ref
_FindCondition );
```

```
        }
        #endregion // 設定の保存と読み出し

        #region 点字ディスプレイの操作
        /// <summary>
        /// 点字ディスプレイの使用を開始
        /// </summary>
        public void BeginUsingBraille()
        {
            if( NbsEngine.Instance == null )
            {
                string message = _Localizer.TryGetString(
                    "AiBEditForm.Msg_NBSEngineNotInstalled", "NBS Engine is not
installed." );
                NbsMessageBox.Show( _View, message, MessageBoxButtons.OK,
MessageBoxIcon.Error );
                _View.UsingBraille = false;
                return;
            }

            try
            {
                // 点字ディスプレイの使用を開始
                NbsEngine.Instance.AttachToServer( _DeviceConfig );
                NbsEngine.Instance.Lock();

                // 点字の詳細な表示設定を行う
                NbsEngine.Instance.BrailleConfig = _BrailleConfig;

                // リーダーを初期化
                _TextBoxBrailler = new RichEdit20WBrailler();
                if( ActiveDocument != null && ActiveDocument.TextBox != null )
                    _TextBoxBrailler.ActiveControl =
(RichEdit20W)ActiveDocument.TextBox;

                // 点字ディスプレイの表示を更新
                _TextBoxBrailler.UpdateDisplay();

                _View.UsingBraille = true;
            }
            catch( InitializeFailedException ex )
            {
                string format = _Localizer.TryGetString(
                    "AiBEditForm.Msg_FailedToInitializeBrailleDisplay",
                    "Failed to initialize Braille display. {0}" );
                NbsMessageBox.Show( _View, String.Format(format, ex.Message),
MessageBoxButtons.OK, MessageBoxIcon.Error );
                _View.UsingBraille = false;
            }
        }
```

```
        /// <summary>
        /// 点字ディスプレイの使用を停止
        /// </summary>
        public void EndUsingBraille()
        {
            if( NbsEngine.Instance != null )
            {
                NbsEngine.Instance.ClearDisplay();
                NbsEngine.Instance.DetachFromServer();
            }
            if( _TextBoxBrailler != null )
            {
                _TextBoxBrailler.Dispose();
                _TextBoxBrailler = null;
            }

            _View.UsingBraille = false;

            // End の直後に Begin を呼ばれると WinPin.exe の再起動が間に合わない事があるため、
時間を稼ぐ
            Win32.Sleep( 200 );
        }

        /// <summary>
        /// 点字ディスプレイの機種を設定
        /// </summary>
        public void ConfigureBrailleDevice()
        {
            DeviceConfigForm    prefForm;
            DialogResult        result;

            // デバイス設定ダイアログを表示
            prefForm = new DeviceConfigForm( _DeviceConfig );
            prefForm.StartPosition = FormStartPosition.CenterParent;
            result = prefForm.ShowDialog( _View );
            if( result != DialogResult.OK )
            {
                return; // キャンセルされた
            }

            // 点字エンジンを再起動
            if( _View.UsingBraille )
            {
                EndUsingBraille();
                BeginUsingBraille();
            }
        }

        /// <summary>
```

```
        /// 点字ディスプレイの点字体系を設定
        /// </summary>
        public void ConfigureBrailleSystem()
        {
            BrailleConfigForm   configForm;
            DialogResult        result;

            // 点字表示設定ダイアログを表示
            configForm = new BrailleConfigForm( _BrailleConfig );
            configForm.StartPosition = FormStartPosition.CenterParent;
            result = configForm.ShowDialog( _View );
            if( result != DialogResult.OK )
            {
                return; // キャンセルされた
            }

            // 点字表示設定を更新
            _BrailleConfig = configForm.BrailleConfig;
            if( NbsEngine.Instance != null )
            {
                NbsEngine.Instance.BrailleConfig = _BrailleConfig;
            }
        }
        #endregion // 点字ディスプレイの操作

        #region 検索
        /// <summary>
        /// ダイアログを開いて文字列検索を行う
        /// </summary>
        public void Find()
        {
            DialogResult    result;
            FindDialog      findForm;
            FindCondition   condition = new FindCondition( _FindCondition );

            // if nothing selected, use word at caret as initial seach pattern.
            // or something selected, use selected text instead.
            if( ActiveDocument.TextBox.SelectedText.Length == 0 )
            {
                condition.Pattern = ActiveDocument.TextBox.GetWordAt(
ActiveDocument.TextBox.SelectionStart );
                if( condition.Pattern == null )
                {
                    condition.Pattern = "";
                }
            }
            else
            {
                condition.Pattern = ActiveDocument.TextBox.SelectedText;
            }
```

```
            using( findForm = new FindDialog(ref condition) )
            {
                // ask find condition
                result = findForm.ShowDialog( _View );
                if( result == DialogResult.Cancel )
                {
                    return;
                }
                _FindCondition = condition;

                // start search
                if( findForm.FindBackward )
                {
                    FindPrevious();
                }
                else
                {
                    FindNext();
                }
            }
        }

        /// <summary>
        /// 直前の検索条件で後方検索を行う
        /// </summary>
        public void FindNext()
        {
            if( _FindCondition.Pattern == String.Empty )
            {
                Win32.MessageBeep_Notify();
                return;
            }
            int matchIndex;
            int matchLength;
            bool found;

            // find the pattern
            found = FindLogic.FindNext( ActiveDocument.TextBox, _FindCondition, out
matchIndex, out matchLength );
            if( found != true )
            {
                Win32.MessageBeep_Notify();
                return;
            }

            // select the pattern
            ActiveDocument.TextBox.Select( matchIndex, matchLength );
        }
```

```
        /// <summary>
        /// 直前の検索条件で前方検索を行う
        /// </summary>
        public void FindPrevious()
        {
            if( _FindCondition.Pattern == String.Empty )
            {
                Win32.MessageBeep_Notify();
                return;
            }
            int matchIndex;
            int matchLength;
            bool found;

            // find the pattern
            found = FindLogic.FindPrevious( ActiveDocument.TextBox, _FindCondition, out
matchIndex, out matchLength );
            if( found != true )
            {
                Win32.MessageBeep_Notify();
                return;
            }

            // select the pattern
            ActiveDocument.TextBox.Select( matchIndex, matchLength );
        }

        /// <summary>
        /// ダイアログを表示してユーザが指定した行にキャレットを移動
        /// </summary>
        public void GoToLine()
        {
            GotoLineDialog  dialog;
            DialogResult    result;
            int             line, col;

            if( ActiveDocument == null )
            {
                Win32.MessageBeep_Notify();
                return;
            }

            // get current caret position
            ActiveDocument.TextBox.GetCaretIndex( out line, out col );

            // show dialog to ask user where the caret should be moved to
            using( dialog = new GotoLineDialog() )
            {
                dialog.StartPosition = FormStartPosition.CenterParent;
                dialog.LineNumber = line + 1;
```

```
            // show dialog
            result = dialog.ShowDialog( _View );
            if( result != DialogResult.OK )
            {
                return;
            }

            // move there
            ActiveDocument.TextBox.SetCaretIndex( dialog.LineNumber - 1, 0 );
        }
    }

    /// <summary>
    /// キャレット位置にあるカッコに対応したカッコの位置にキャレットを移動
    /// </summary>
    public void GoToMatchedBracket()
    {
        int     line, col;
        string  message;
        string  format;
        int caretIndex;
        int pairIndex;

        // find pair and go there
        caretIndex = ActiveDocument.TextBox.GetCaretIndex();
        pairIndex = FindLogic.FindMatchedBracket( ActiveDocument.TextBox.Text,
caretIndex );
        if( pairIndex == -1 )
        {
            Win32.MessageBeep_Notify();
            return;
        }
        ActiveDocument.TextBox.SetCaretIndex( pairIndex );

        // make message
        ActiveDocument.TextBox.GetCaretIndex( out line, out col );
        format = _Localizer.TryGetString( "AiBEditForm.Msg_SayCurrentLine", "line
{0}  column {1}" );
        message = String.Format( format, line+1, col+1 );

        // say message
        if( AutoSpeaker.IsAlive() )
        {
            AutoSpeaker.Instance.Speak( message );
        }
    }
    #endregion

    public void SayCaretPosition()
```

```
        {
            int     line, col;
            string  message;
            string  format;

            // make message
            ActiveDocument.TextBox.GetCaretIndex( out line, out col );
            format = _Localizer.TryGetString( "AiBEditForm.Msg_SayCurrentLine", "line
{0}  column {1}" );
            message = String.Format( format, line+1, col+1 );

            // say message
            if( AutoSpeaker.IsAlive() )
            {
                AutoSpeaker.Instance.Speak( message );
            }
            else
            {
                NbsMessageBox.Show( _View, message );
            }
        }

        /// <summary>
        /// ファイルシステムから正確なパス文字列を再取得する。
        /// </summary>
        /// <param name="filePath">正確なパスを再取得したいファイルのパス</param>
        /// <returns>再取得したパス</returns>
        /// <remarks>
        /// Windows では大文字小文字を区別すると不正なパスでも使えてしまう。
        /// 大文字小文字の別も正確に取得するために呼ぶ。
        /// </remarks>
        static string ReGetFilePath( string filePath )
        {
            string dir = Path.GetDirectoryName( filePath );
            dir = Path.GetFullPath( dir );
            string filter = Path.GetFileName( filePath );
            return Directory.GetFiles( dir, filter )[0];
        }
    }
}
```

AiBEditForm.cs

```
// file     : AiBEditForm.cs
// brief    : main form
// update   : 2007-02-25
//=========================================================
//#define  ACCEDIT_IDE
using System;
using System.IO;
using System.Collections;
```

```
using System.Text;
using System.Text.RegularExpressions;
using System.Runtime.InteropServices;
using System.Windows.Forms;

namespace Sgry.AiBTools.AiBEdit
{
    using AiBTools;
    using Nbs;
    using Nbs.Forms;

    internal class AiBEditForm : Form
    {
        #region Constants
        const int WM_USER       = 0x0400;
        /// <summary>
        /// カーソルを移動するメッセージ
        /// </summary>
        /// <remarks>
        /// LPARAM で移動先の行、
        /// LPARAM で移動先の桁を指定します。
        /// 単位は文字単位であり、1始まりのインデックスで指定します。
        /// </remarks>
        public const int AIBE_MOVECARET = WM_USER + 1;

        /// <summary>
        /// 外部プロセスから書類を開かせるために使うメッセージ
        /// </summary>
        /// <remarks>
        /// このメッセージを受け取ったプロセスは、
        /// 共有ファイルの先頭行にあるパスを開きます。
        /// </remarks>
        public const int AIBE_OPENFILE = WM_USER + 2;
        #endregion

        #region フィールド
        AppLogic            _AppLogic;
        bool                _Shown = false;
//      ListViewReader      _ErrorListReader;
        public static Localizer Localizer = new Localizer();
        #endregion

        #region UI アクセス
        /// <summary>
        /// 「最近使ったファイル」メニュー
        /// </summary>
        public MenuItem MruMenu
        {
            get{ return _MI_File_Mru; }
        }
```

```
/// <summary>
/// 「編集モード」サブメニュー
/// </summary>
public MenuItem EditModeMenu
{
    get{ return _MI_Tool_Mode; }
}

/// <summary>
/// 「ウィンドウ」メニュー
/// </summary>
public MenuItem WindowMenu
{
    get{ return _MI_Window; }
}

/// <summary>
/// タブコントロール
/// </summary>
public AiBTabControl TabControl
{
    get{ return _TabControl; }
}

/// <summary>
/// テキストボックスが配置されるパネル
/// </summary>
public Panel TextBoxPanel
{
    get{ return _TextBoxPanel; }
}

/// <summary>
/// 点字ディスプレイを利用しているかどうかを表すフラグ
/// </summary>
public bool UsingBraille
{
    get{ return _MI_Braille_Use.Checked; }
    set{ _MI_Braille_Use.Checked = value; }
}

/// <summary>
/// 編集関連のメニュー項目の有効無効を切り替える
/// </summary>
/// <param name="enable">有効にするか無効にするかを指定します。</param>
/// <param name="isReadonly">読み出し専用モードの場合は true にします。</param>
public void EnableEditMenuItems( bool enable, bool isReadonly )
{
    _MI_File_Close.Enabled = _MI_File_Save.Enabled = _MI_File_SaveAs.Enabled
```

```
                = _MI_File_Reload.Enabled = _MI_Speech_CurrentLine.Enabled
                = _MI_Edit_Undo.Enabled = _MI_Edit_Cut.Enabled = _MI_Edit_Copy.Enabled
                = _MI_Edit_Paste.Enabled = _MI_Edit_Find.Enabled =
_MI_Edit_FindNext.Enabled
                = _MI_Edit_FindPrevious.Enabled = _MI_Edit_SelectAll.Enabled
                = _MI_Edit_JumpToLine.Enabled = _MI_Edit_MatchedBracket.Enabled
                = _MI_Tool_Mode.Enabled = _MI_Tool_Outline.Enabled =
_MI_Tool_Compile.Enabled
                = _MI_Tool_Build.Enabled = _MI_Tool_ErrorList.Enabled
                = enable;
        }

        public void EnableWindowSwitchMenuItems( bool enable )
        {
            _MI_Window_Next.Enabled
                = _MI_Window_Previous.Enabled
                = _MI_Window_List.Enabled
                = enable;
        }

        public void EnableSaveReloadMenuItem( bool enable )
        {
            _MI_File_Save.Enabled = _MI_File_Reload.Enabled = enable;
        }
        #endregion

        #region Init / Dispose
        /// <summary>
        /// 構築
        /// </summary>
        /// <param name="app">アプリケーションオブジェクト</param>
        public AiBEditForm( AppLogic app )
        {
            // initialize components
            InitializeComponents();
            LocalizeComponents();

            Activated += new EventHandler( Form_Activated );

            EnableEditMenuItems( false, false );
            EnableWindowSwitchMenuItems( false );

            _AppLogic = app;
        }

        void Form_Activated( object sender, EventArgs e )
        {
            if( _Shown == false )
            {
                _Shown = true;
```

```
        if( JawsSpeaker.IsAlive() )
        {
            JawsSpeaker.Instance.Stop();
        }
    }
}

/// <summary>
/// フォームが作成された直後の動作。
/// 点字ディスプレイサーバへの接続、キャレット位置へのスクロールなど。
/// </summary>
protected override void OnLoad( EventArgs e )
{
    base.OnLoad( e );

    // 点字ディスプレイを使う設定が残っていれば、使用開始
    if( _MI_Braille_Use.Checked )
    {
        _AppLogic.BeginUsingBraille();
    }
}
#endregion // Init / Dispose

#region 親クラスの動作の変更
/// <summary>
/// フォームのキャプションテキストを更新
/// </summary>
public override void ResetText()
{
    StringBuilder newText = new StringBuilder();

    // 書類名とアプリケーション名を結合
    if( _AppLogic.ActiveDocument != null )
    {
        newText.Append( _AppLogic.ActiveDocument.Title );
        newText.Append( " - " );
    }
    newText.Append( "AiB Edit" );

    // 設定
    Text = newText.ToString();
}
#endregion

#region メニューコマンド
//=====================================================
// File menu
void _MI_File_New_Click( object sender, EventArgs e )
{
    _AppLogic.CreateUntitledDocument();
```

```
    ResetText();
}
void _MI_File_Open_Click( object sender, EventArgs e )
{
    _AppLogic.OpenDocument();
}
void _MI_File_Save_Click( object sender, EventArgs e )
{
    _AppLogic.SaveDocument();
}
void _MI_File_SaveAs_Click( object sender, EventArgs e )
{
    _AppLogic.SaveDocumentAs();
}
void _MI_File_Reload_Same_Click( object sender, EventArgs e )
{
    _AppLogic.ReloadDocument();
}
void _MI_File_Reload_Sjis_Click( object sender, EventArgs e )
{
    _AppLogic.ReloadDocument( Encoding.GetEncoding("Shift_JIS"), false );
}
void _MI_File_Reload_Jis_Click( object sender, EventArgs e )
{
    _AppLogic.ReloadDocument( Encoding.GetEncoding("iso-2022-jp"), false  );
}
void _MI_File_Reload_Euc_Click( object sender, EventArgs e )
{
    _AppLogic.ReloadDocument( Encoding.GetEncoding("EUC-JP"), false  );
}
void _MI_File_Reload_Utf8_Click( object sender, EventArgs e )
{
    _AppLogic.ReloadDocument( Encoding.UTF8, false );
}
//-------------------------------------------------------
void _MI_File_Close_Click(object sender, EventArgs e)
{
    Win32.LockWindowUpdate( Handle );
    _AppLogic.CloseDocument();
    Win32.LockWindowUpdate( IntPtr.Zero );
}
void _MI_File_Exit_Click(object sender, EventArgs e)
{
    this.Close();
}

//=======================================================
// Edit menu
void _MI_Edit_Undo_Click(object sender, EventArgs e)
{
```

```
    _AppLogic.ActiveDocument.TextBox.Undo();
}
//-----------------------------------------------------
void _MI_Edit_Cut_Click( object sender, EventArgs e )
{
    _AppLogic.ActiveDocument.TextBox.Cut();
}
void _MI_Edit_Copy_Click( object sender, EventArgs e )
{
    _AppLogic.ActiveDocument.TextBox.Copy();
}
void _MI_Edit_Paste_Click( object sender, EventArgs e )
{
    _AppLogic.ActiveDocument.TextBox.Paste();
}
//-----------------------------------------------------
void _MI_Edit_Find_Click( object sender, EventArgs e )
{
    _AppLogic.Find();
}
void _MI_Edit_FindNext_Click( object sender, EventArgs e )
{
    _AppLogic.FindNext();
}
void _MI_Edit_FindPrevious_Click( object sender, EventArgs e )
{
    _AppLogic.FindPrevious();
}
//-----------------------------------------------------
void _MI_Edit_SelectAll_Click( object sender, EventArgs e )
{
    _AppLogic.ActiveDocument.TextBox.SelectAll();
}
void _MI_Edit_GoToLine_Click( object sender, EventArgs e )
{
    _AppLogic.GoToLine();
}
void _MI_Edit_MatchedBracket_Click( object sender, EventArgs e )
{
    _AppLogic.GoToMatchedBracket();
}

//=====================================================
// Speech menu
void _MI_Speech_CurrentLine_Click( object sender, EventArgs e )
{
    _AppLogic.SayCaretPosition();
}

//=====================================================
```

```
        // Tool menu
        void _MI_Tool_Outline_Click( object sender, EventArgs e )
        {
            _AppLogic.ShowOutlineDialog();
        }
#if ACCEDIT_IDE
        void _MI_Tool_Compile_Click( object sender, EventArgs e )
        {
            DoCompilation( false );
        }
        void _MI_Tool_Build_Click( object sender, EventArgs e )
        {
            DoCompilation( true );
        }
        void _MI_Tool_ErrorList_Click( object sender, EventArgs e )
        {
            // 何も選択されていなければ最初の項目を選択
            if( 0 < _ErrorListView.Items.Count
                && _ErrorListView.SelectedIndices.Count == 0 )
            {
                _ErrorListView.Items[0].Focused = true;
                _ErrorListView.Items[0].Selected = true;
            }
            _ErrorListView.Focus();
        }
#endif

        //=====================================================
        // Window menu
        void _MI_Window_Next_Click( object sender, EventArgs e )
        {
            Win32.LockWindowUpdate( Handle );
            _TabControl.SelectNextTab();
            Win32.LockWindowUpdate( IntPtr.Zero );

            AutoSpeaker.Instance.Speak( _TabControl.SelectedItem.ToString() );
        }
        void _MI_Window_Previous_Click( object sender, EventArgs e )
        {
            Win32.LockWindowUpdate( Handle );
            _TabControl.SelectPreviousTab();
            Win32.LockWindowUpdate( IntPtr.Zero );

            AutoSpeaker.Instance.Speak( _TabControl.SelectedItem.ToString() );
        }
        void _MI_Window_List_Click( object sender, EventArgs e )
        {
            _AppLogic.ShowDocumentList();
        }
```

```
//=========================================================
// Braille display menu
void _MI_Braille_Use_Click( object sender, EventArgs e )
{
    // チェックがついていなければ、点字ディスプレイの使用を開始
    if( UsingBraille )
    {
        _AppLogic.EndUsingBraille();
    }
    else
    {
        _AppLogic.BeginUsingBraille();
    }
}
void _MI_Braille_DeviceConfig_Click( object sender, EventArgs e )
{
    _AppLogic.ConfigureBrailleDevice();
}
void _MI_Braille_BrailleConfig_Click( object sender, EventArgs e )
{
    _AppLogic.ConfigureBrailleSystem();
}
#endregion // メニューコマンド

#region ドラッグ&ドロップ
/// <summary>
/// オブジェクトをドラッグしながらカーソルがフォーム上に入ってきた時の動作
/// </summary>
void Form_DragEnter( object sender, DragEventArgs e )
{
    object      enteringData;
    string[]    enteringFilePaths;

    // ファイル（ディレクトリ）のドロップか確認
    enteringData = e.Data.GetData( DataFormats.FileDrop );
    if( enteringData == null )
    {
        return;
    }

    // ドロップされたファイルのパスを取得
    enteringFilePaths = (string[])enteringData;
    if( enteringFilePaths.Length < 1 )
    {
        return;
    }

    // ディレクトリなら拒否する
    if( Directory.Exists(enteringFilePaths[0]) )
    {
```

```
            return;
        }

        e.Effect = DragDropEffects.Copy;
    }

    /// <summary>
    /// オブジェクトがドロップされた時の動作
    /// </summary>
    void Form_DragDrop( object sender, DragEventArgs e )
    {
        try
        {
            object      droppedData;
            string[]    droppedFilePaths;

            // ドロップされたファイル群のパスを取得
            droppedData = e.Data.GetData( DataFormats.FileDrop );
            if( droppedData == null )
            {
                return;
            }

            droppedFilePaths = (string[])droppedData;
            if( droppedFilePaths.Length < 1 )
            {
                return;
            }

            // ファイルを開く
            _AppLogic.OpenDocument( droppedFilePaths[0] );
        }
        catch( Exception )
        {}
    }
    #endregion // ドラッグ&ドロップ

    #region ウィンドウメッセージの処理
    /// <summary>
    /// ショートカットキーの動作を変更
    /// </summary>
    protected override bool ProcessCmdKey( ref Message msg, Keys keyData )
    {
        const int VK_OEM4 = 219;
        const int VK_OEM6 = 221;

        int key = msg.WParam.ToInt32();
        if( Win32.IsControlKeyDown() )
        {
            if( key == VK_OEM4 || key == VK_OEM6 )
```

```
            {
                _AppLogic.GoToMatchedBracket();
                return true;
            }
            else if( (Keys)key == Keys.Tab )
            {
                if( Win32.IsShiftKeyDown() )
                    _MI_Window_Previous_Click( this, EventArgs.Empty );
                else
                    _MI_Window_Next_Click( this, EventArgs.Empty );
                return true;
            }
        }
        //DEBUG-->
        if( keyData == (Keys.Control | Keys.Cancel) ) // Ctrl+Pause (Break key)
        {
            GC.Collect();
            Win32.MessageBeep_Notify();
        }
        //<-- DEBUG

        return base.ProcessCmdKey( ref msg, keyData );
    }

    /// <summary>
    /// Windows メッセージを処理するウィンドウプロシージャ
    /// </summary>
    /// <param name="m">Windows メッセージ</param>
    protected override void WndProc( ref Message m )
    {
        // キャレット移動？
        if( m.Msg == AIBE_MOVECARET )
        {
            int lineIndex = m.WParam.ToInt32() - 1;
            int columnIndex = m.LParam.ToInt32() - 1;

            _AppLogic.ActiveDocument.TextBox.SetCaretIndex( lineIndex, columnIndex
);
        }
        // ファイルを開く？
        else if( m.Msg == AIBE_OPENFILE )
        {
            string  filePath = SharedData.GetLastLine();

            _AppLogic.OpenDocument( filePath );
            // （このままではウィンドウが切り替わった事が分かりにくい）
        }

        base.WndProc( ref m );
    }
```

```
        #endregion

        #region エラーリストビュー関係
#if ACCEDIT_IDE
        void _ErrorListView_KeyDown( object sender, KeyEventArgs e )
        {
            int focusedItemIndex;

            // エスケープなら、テキストボックスにフォーカスを移動
            if( e.KeyCode == Keys.Escape )
            {
                _AppLogic.ActiveDocument.TextBox.Focus();
            }

            // リスト上端で上を、下端で下に移動しようとしたら、通知
            focusedItemIndex = _ErrorListView.FocusedItem.Index;
            if( e.KeyCode == Keys.Up && focusedItemIndex == 0 )
            {
                Win32.MessageBeep_Notify();
            }
            else if( e.KeyCode == Keys.Down && focusedItemIndex ==
_ErrorListView.Items.Count-1 )
            {
                Win32.MessageBeep_Notify();
            }
        }

        void _ErrorListView_ItemActivate( object sender, EventArgs e )
        {
            try
            {
                int lineIndex = Int32.Parse(
_ErrorListView.FocusedItem.SubItems[2].Text ) - 1;
                int hwColumnIndex = Int32.Parse(
_ErrorListView.FocusedItem.SubItems[3].Text ) - 1;
                int columnIndex = Utl.CalcCharIndexFromHalfWidthIndex(
                                    _AppLogic.ActiveDocument.TextBox.GetLine(lineIndex),
hwColumnIndex, 8 );

                Win32.LockWindowUpdate( _AppLogic.ActiveDocument.TextBox.Handle );
                _AppLogic.ActiveDocument.TextBox.Focus();
                _AppLogic.ActiveDocument.TextBox.SetCaretIndex( lineIndex, columnIndex
);
                Win32.LockWindowUpdate( IntPtr.Zero );
            }
            catch( FormatException )
            {
                Win32.MessageBeep_Notify();
            }
        }
```

```
#endif
        #endregion // エラーリストビュー関係

#if ACCEDIT_IDE
        void DoCompilation( bool doBuild )
        {
            NowCompilingDialog  dialog;

            // ファイルが保存されていなければ保存
            if( _AppLogic.ActiveDocument.Path == String.Empty )
            {
                DialogResult dlgResult = _AppLogic.SaveDocument();
                if( dlgResult != DialogResult.OK )
                {
                    return; // 保存できなかったのでコンパイルは中止
                }
            }

            // コンパイル実行
            _ErrorListView.Items.Clear();
            dialog = new NowCompilingDialog( doBuild );
            dialog.Compile( this, _AppLogic.ActiveDocument.FileType,
_AppLogic.ActiveDocument );

            // コンパイルに成功したら、メッセージを出してエディタに戻る
            if( dialog.Result == CompileResult.Succeeded )
            {
                string message = Localizer.TryGetString( "Msg_SuccessfullyCompiled",
"Successfully compiled." );
                NbsMessageBox.Show( this, message );
                _ErrorListView.Items.Add( _DummyListViewItem );
                _AppLogic.ActiveDocument.TextBox.Focus();
            }
            // コンパイルエラーが発生したら、エラーリストビューにフォーカス
            else if( dialog.Result == CompileResult.CompileErrorOccured )
            {
                // もしコンパイルエラーを一つも検出できなかった場合はダミーを表示
                if( _ErrorListView.Items.Count == 0 )
                {
                    _ErrorListView.Items.Add( _DummyListViewItem );
                }
                _MI_Tool_ErrorList_Click( this, EventArgs.Empty );
            }
            // コンパイルの実行に失敗した
            else
            {
                string message = Localizer.TryGetString(
"Msg_FailedToExecuteCompilation", "failed to execute compilation" );
                NbsMessageBox.Show( this, message, MessageBoxButtons.OK,
MessageBoxIcon.Error );
```

```
            }
        }
#endif

        #region UI Component Initialization
        /// <summary>
        /// 文字列をローカライズ
        /// </summary>
        void LocalizeComponents()
        {
            Localizer.LoadResourceFile();

            Localizer.TryGetString( "AiBEditForm._MI_File.Text", _MI_File );
            Localizer.TryGetString( "AiBEditForm._MI_File_New.Text", _MI_File_New );
            Localizer.TryGetString( "AiBEditForm._MI_File_Open.Text", _MI_File_Open );
            Localizer.TryGetString( "AiBEditForm._MI_File_Save.Text", _MI_File_Save );
            Localizer.TryGetString( "AiBEditForm._MI_File_SaveAs.Text", _MI_File_SaveAs
);
            Localizer.TryGetString( "AiBEditForm._MI_File_Reload.Text", _MI_File_Reload
);
            Localizer.TryGetString( "AiBEditForm._MI_File_Reload_AnalyzeAgain.Text",
_MI_File_Reload_AnalyzeAgain );
            Localizer.TryGetString( "AiBEditForm._MI_File_Reload_Sjis.Text",
_MI_File_Reload_Sjis );
            Localizer.TryGetString( "AiBEditForm._MI_File_Reload_Euc.Text",
_MI_File_Reload_Euc );
            Localizer.TryGetString( "AiBEditForm._MI_File_Reload_Jis.Text",
_MI_File_Reload_Jis );
            Localizer.TryGetString( "AiBEditForm._MI_File_Reload_Utf8.Text",
_MI_File_Reload_Utf8 );
            Localizer.TryGetString( "AiBEditForm._MI_File_Close.Text", _MI_File_Close
);
            Localizer.TryGetString( "AiBEditForm._MI_File_Mru.Text", _MI_File_Mru );
            Localizer.TryGetString( "AiBEditForm._MI_File_Exit.Text", _MI_File_Exit );

            Localizer.TryGetString( "AiBEditForm._MI_Edit.Text", _MI_Edit );
            Localizer.TryGetString( "AiBEditForm._MI_Edit_Undo.Text", _MI_Edit_Undo );
            Localizer.TryGetString( "AiBEditForm._MI_Edit_Cut.Text", _MI_Edit_Cut );
            Localizer.TryGetString( "AiBEditForm._MI_Edit_Copy.Text", _MI_Edit_Copy );
            Localizer.TryGetString( "AiBEditForm._MI_Edit_Paste.Text", _MI_Edit_Paste
);
            Localizer.TryGetString( "AiBEditForm._MI_Edit_Find.Text", _MI_Edit_Find );
            Localizer.TryGetString( "AiBEditForm._MI_Edit_FindNext.Text",
_MI_Edit_FindNext );
            Localizer.TryGetString( "AiBEditForm._MI_Edit_FindPrevious.Text",
_MI_Edit_FindPrevious );
            Localizer.TryGetString( "AiBEditForm._MI_Edit_SelectAll.Text",
_MI_Edit_SelectAll );
            Localizer.TryGetString( "AiBEditForm._MI_Edit_JumpToLine.Text",
_MI_Edit_JumpToLine );
```

```
            Localizer.TryGetString( "AiBEditForm._MI_Edit_MatchedBracket.Text",
_MI_Edit_MatchedBracket );

            Localizer.TryGetString( "AiBEditForm._MI_Speech.Text", _MI_Speech );
            Localizer.TryGetString( "AiBEditForm._MI_Speech_CurrentLine.Text",
_MI_Speech_CurrentLine );

            Localizer.TryGetString( "AiBEditForm._MI_Tool.Text", _MI_Tool );
            Localizer.TryGetString( "AiBEditForm._MI_Tool_Mode.Text", _MI_Tool_Mode );
            Localizer.TryGetString( "AiBEditForm._MI_Tool_Outline.Text",
_MI_Tool_Outline );
            Localizer.TryGetString( "AiBEditForm._MI_Tool_Compile.Text",
_MI_Tool_Compile );
            Localizer.TryGetString( "AiBEditForm._MI_Tool_ErrorList.Text",
_MI_Tool_ErrorList );

            Localizer.TryGetString( "AiBEditForm._MI_Window.Text", _MI_Window );
            Localizer.TryGetString( "AiBEditForm._MI_Window_Next.Text", _MI_Window_Next
);
            Localizer.TryGetString( "AiBEditForm._MI_Window_Previous.Text",
_MI_Window_Previous );
            Localizer.TryGetString( "AiBEditForm._MI_Window_List.Text", _MI_Window_List
);

            Localizer.TryGetString( "AiBEditForm._MI_Braille.Text", _MI_Braille );
            Localizer.TryGetString( "AiBEditForm._MI_Braille_Use.Text", _MI_Braille_Use
);
            Localizer.TryGetString( "AiBEditForm._MI_Braille_BrailleConfig.Text",
_MI_Braille_BrailleConfig );
            Localizer.TryGetString( "AiBEditForm._MI_Braille_DeviceConfig.Text",
_MI_Braille_DeviceConfig );
#if ACCEDIT_IDE
            Localizer.TryGetString( "AiBEditForm._ErrorListView.Columns[0].Text",
_ErrorListView.Columns[0] );
            Localizer.TryGetString( "AiBEditForm._ErrorListView.Columns[1].Text",
_ErrorListView.Columns[1] );
            Localizer.TryGetString( "AiBEditForm._ErrorListView.Columns[2].Text",
_ErrorListView.Columns[2] );
            Localizer.TryGetString( "AiBEditForm._DummyListViewItem.SubItems[0]",
_DummyListViewItem.SubItems[0] );
#endif
        }

        /// <summary>
        /// UI 部品を初期化します。
        /// </summary>
        /// <remarks>
        /// Visual Studio のビジュアルエディタとは互換性が無いので注意。
        /// </remarks>
        void InitializeComponents()
```

```
        {
            SuspendLayout();
            //
            // _MainMenu
            //
            _MainMenu.MenuItems.AddRange(
                    new MenuItem[] {_MI_File, _MI_Edit, _MI_Speech, _MI_Tool, _MI_Window,
_MI_Braille}
                );
            //
            // _MI_File
            //
            _MI_File.Index = 0;
            _MI_File.MenuItems.AddRange(
                    new MenuItem[] {_MI_File_New, _MI_File_Open, _MI_File_Save,
_MI_File_SaveAs, _MI_File_Separator,
                        _MI_File_Reload, _MI_File_Separator2,
                        _MI_File_Mru, _MI_File_Separator3,
                        _MI_File_Close, _MI_File_Exit}
                );
            _MI_File.Text = "&File";
            //
            // _MI_File_New
            //
            _MI_File_New.Index = 0;
            _MI_File_New.Shortcut = Shortcut.CtrlN;
            _MI_File_New.Text = "&New...";
            _MI_File_New.Click += new EventHandler(_MI_File_New_Click);
            //
            // _MI_File_Open
            //
            _MI_File_Open.Index = 1;
            _MI_File_Open.Shortcut = Shortcut.CtrlO;
            _MI_File_Open.Text = "&Open...";
            _MI_File_Open.Click += new EventHandler(_MI_File_Open_Click);
            //
            // _MI_File_Save
            //
            _MI_File_Save.Index = 2;
            _MI_File_Save.Shortcut = Shortcut.CtrlS;
            _MI_File_Save.Text = "&Save";
            _MI_File_Save.Click += new EventHandler(_MI_File_Save_Click);
            //
            // _MI_File_SaveAs
            //
            _MI_File_SaveAs.Index = 3;
            _MI_File_SaveAs.Shortcut = Shortcut.CtrlShiftS;
            _MI_File_SaveAs.Text = "Save &as...";
            _MI_File_SaveAs.Click += new EventHandler(_MI_File_SaveAs_Click);
            //
```

```
// _MI_File_Separator
//
_MI_File_Separator.Index = 4;
_MI_File_Separator.Text = "-";
//
// _MI_File_Reload
//
_MI_File_Reload.Index = 5;
_MI_File_Reload.Text = "&Reload";
_MI_File_Reload.MenuItems.AddRange(
        new MenuItem[] {
            _MI_File_Reload_AnalyzeAgain,
            _MI_File_Reload_Separator,
            _MI_File_Reload_Sjis,
            _MI_File_Reload_Jis,
            _MI_File_Reload_Euc,
            _MI_File_Reload_Utf8,
        }
    );
//
// _MI_File_Reload_AnalyzeAgain
//
_MI_File_Reload_AnalyzeAgain.Index = 0;
_MI_File_Reload_AnalyzeAgain.Text = "Analyze encoding a&gain";
_MI_File_Reload_AnalyzeAgain.Click
    += new EventHandler( _MI_File_Reload_Same_Click );
//
// _MI_File_Reload_Separator
//
_MI_File_Reload_Separator.Index = 1;
_MI_File_Reload_Separator.Text = "-";
//
// _MI_File_Reload_Sjis
//
_MI_File_Reload_Sjis.Index = 2;
_MI_File_Reload_Sjis.Text = "as &Shift JIS";
_MI_File_Reload_Sjis.Click
    += new EventHandler( _MI_File_Reload_Sjis_Click );
//
// _MI_File_Reload_Jis
//
_MI_File_Reload_Jis.Index = 3;
_MI_File_Reload_Jis.Text = "as &JIS";
_MI_File_Reload_Jis.Click += new EventHandler( _MI_File_Reload_Jis_Click );
//
// _MI_File_Reload_Euc
//
_MI_File_Reload_Euc.Index = 4;
_MI_File_Reload_Euc.Text = "as &EUC-JP";
_MI_File_Reload_Euc.Click += new EventHandler( _MI_File_Reload_Euc_Click );
```

```
//
// _MI_File_Reload_Utf8
//
_MI_File_Reload_Utf8.Index = 5;
_MI_File_Reload_Utf8.Text = "as UTF-&8";
_MI_File_Reload_Utf8.Click
    += new EventHandler( _MI_File_Reload_Utf8_Click );
//
// _MI_File_Separator2
//
_MI_File_Separator2.Index = 6;
_MI_File_Separator2.Text = "-";
//
// _MI_File_Mru
//
_MI_File_Mru.Index = 7;
_MI_File_Mru.Text = "Recent &files";
_MI_File_Mru.Enabled = false;
//
// _MI_File_Separator3
//
_MI_File_Separator3.Index = 8;
_MI_File_Separator3.Text = "-";
//
// _MI_File_Close
//
_MI_File_Close.Index = 9;
_MI_File_Close.Shortcut = Shortcut.CtrlW;
_MI_File_Close.Text = "&Close";
_MI_File_Close.Click += new EventHandler(_MI_File_Close_Click);
//
// _MI_File_Exit
//
_MI_File_Exit.Index = 10;
_MI_File_Exit.Shortcut = Shortcut.CtrlQ;
_MI_File_Exit.Text = "E&xit";
_MI_File_Exit.Click += new EventHandler( _MI_File_Exit_Click );
//
// _MI_Edit
//
_MI_Edit.Index = 1;
_MI_Edit.MenuItems.AddRange(
        new MenuItem[] {
            _MI_Edit_Undo,
            _MI_Edit_Separator1,
            _MI_Edit_Cut, _MI_Edit_Copy, _MI_Edit_Paste,
            _MI_Edit_Separator2,
            _MI_Edit_Find, _MI_Edit_FindNext, _MI_Edit_FindPrevious,
            _MI_Edit_Separator3,
            _MI_Edit_SelectAll, _MI_Edit_JumpToLine, _MI_Edit_MatchedBracket
```

```
        }
    );
_MI_Edit.Text = "&Edit";
//
// _MI_Edit_Undo
//
_MI_Edit_Undo.Index = 0;
_MI_Edit_Undo.Shortcut = Shortcut.CtrlZ;
_MI_Edit_Undo.Text = "&Undo";
_MI_Edit_Undo.Click += new EventHandler(_MI_Edit_Undo_Click);
//
// _MI_Edit_Separator1
//
_MI_Edit_Separator1.Index = 1;
_MI_Edit_Separator1.Text = "-";
//
// _MI_Edit_Cut
//
_MI_Edit_Cut.Index = 2;
_MI_Edit_Cut.Shortcut = Shortcut.CtrlX;
_MI_Edit_Cut.Text = "Cu&t";
_MI_Edit_Cut.Click += new EventHandler(_MI_Edit_Cut_Click);
//
// _MI_Edit_Copy
//
_MI_Edit_Copy.Index = 3;
_MI_Edit_Copy.Shortcut = Shortcut.CtrlC;
_MI_Edit_Copy.Text = "&Copy";
_MI_Edit_Copy.Click += new EventHandler(_MI_Edit_Copy_Click);
//
// _MI_Edit_Paste
//
_MI_Edit_Paste.Index = 4;
_MI_Edit_Paste.Shortcut = Shortcut.CtrlV;
_MI_Edit_Paste.Text = "&Paste";
_MI_Edit_Paste.Click += new EventHandler(_MI_Edit_Paste_Click);
//
// _MI_Edit_Separator2
//
_MI_Edit_Separator2.Index = 5;
_MI_Edit_Separator2.Text = "-";
//
// _MI_Edit_Find
//
_MI_Edit_Find.Index = 6;
_MI_Edit_Find.Shortcut = Shortcut.CtrlF;
_MI_Edit_Find.Text = "&Find...";
_MI_Edit_Find.Click += new EventHandler(_MI_Edit_Find_Click);
//
// _MI_Edit_FindNext
```

```
//
_MI_Edit_FindNext.Index = 7;
_MI_Edit_FindNext.Shortcut = Shortcut.F3;
_MI_Edit_FindNext.Text = "Find &next";
_MI_Edit_FindNext.Click += new EventHandler(_MI_Edit_FindNext_Click);
//
// _MI_Edit_FindPrevious
//
_MI_Edit_FindPrevious.Index = 8;
_MI_Edit_FindPrevious.Shortcut = Shortcut.ShiftF3;
_MI_Edit_FindPrevious.Text = "Find previou&s";
_MI_Edit_FindPrevious.Click
    += new EventHandler(_MI_Edit_FindPrevious_Click);
//
// _MI_Edit_Separator3
//
_MI_Edit_Separator3.Index = 9;
_MI_Edit_Separator3.Text = "-";
//
// _MI_Edit_SelectAll
//
_MI_Edit_SelectAll.Index = 10;
_MI_Edit_SelectAll.Shortcut = Shortcut.CtrlA;
_MI_Edit_SelectAll.Text = "Select &All";
_MI_Edit_SelectAll.Click += new EventHandler(_MI_Edit_SelectAll_Click);
//
// _MI_Edit_JumpToLine
//
_MI_Edit_JumpToLine.Index = 11;
_MI_Edit_JumpToLine.Shortcut = Shortcut.CtrlG;
_MI_Edit_JumpToLine.Text = "&Go to line...";
_MI_Edit_JumpToLine.Click += new EventHandler(_MI_Edit_GoToLine_Click);
//
// _MI_Edit_MatchedBracket
//
_MI_Edit_MatchedBracket.Index = 12;
_MI_Edit_MatchedBracket.Text = "Go to matched &bracket¥tCtrl+[ / Ctrl+]";
_MI_Edit_MatchedBracket.Click
    += new EventHandler( _MI_Edit_MatchedBracket_Click );
//
// _MI_Speech
//
_MI_Speech.Index = 2;
_MI_Speech.MenuItems.AddRange(
        new MenuItem[] {_MI_Speech_CurrentLine}
    );
_MI_Speech.Text = "&Speech";
//
// _MI_Speech_CurrentLine
//
```

```
        _MI_Speech_CurrentLine.Index = 0;
        _MI_Speech_CurrentLine.Shortcut = Shortcut.CtrlL;
        _MI_Speech_CurrentLine.Text = "Say &caret position...";
        _MI_Speech_CurrentLine.Click
            += new EventHandler(_MI_Speech_CurrentLine_Click);
        //
        // _MI_Tool
        //
        _MI_Tool.Index = 3;
#       if ACCEDIT_IDE
        _MI_Tool.MenuItems.AddRange(
                new MenuItem[] {
                    _MI_Tool_Mode,
                    _MI_Tool_Separator,
                    _MI_Tool_Outline,
                    _MI_Tool_Compile,
                    _MI_Tool_Build,
                    _MI_Tool_ErrorList
                }
            );
#       else
        _MI_Tool.MenuItems.AddRange(
                new MenuItem[] {_MI_Tool_Mode, _MI_Tool_Separator, _MI_Tool_Outline}
            );
#       endif
        _MI_Tool.Text = "&Tools";
        //
        // _MI_Tool_Mode
        //
        _MI_Tool_Mode.Index = 0;
        _MI_Tool_Mode.Text = "Edit &mode";
        //
        // _MI_Tool_Separator
        //
        _MI_Tool_Separator.Index = 1;
        _MI_Tool_Separator.Text = "-";
        //
        // _MI_Tool_Outline
        //
        _MI_Tool_Outline.Index = 2;
        _MI_Tool_Outline.Shortcut = Shortcut.CtrlT;
        _MI_Tool_Outline.Text = "Show ou&tline...";
        _MI_Tool_Outline.Click += new EventHandler(_MI_Tool_Outline_Click);
#       if ACCEDIT_IDE
        //
        // _MI_Tool_Compile
        //
        _MI_Tool_Compile.Index = 3;
        _MI_Tool_Compile.Shortcut = Shortcut.CtrlShiftC;
        _MI_Tool_Compile.Text = "&Compile...";
```

```
        _MI_Tool_Compile.Click += new EventHandler( _MI_Tool_Compile_Click );
        //
        // _MI_Tool_Build
        //
        _MI_Tool_Build.Index = 4;
        _MI_Tool_Build.Shortcut = Shortcut.CtrlShiftB;
        _MI_Tool_Build.Text = "&Build...";
        _MI_Tool_Build.Click += new EventHandler( _MI_Tool_Build_Click );
        //
        // _MI_Tool_ErrorList
        //
        _MI_Tool_ErrorList.Index = 5;
        _MI_Tool_ErrorList.Shortcut = Shortcut.CtrlE;
        _MI_Tool_ErrorList.Text = "Show &error list";
        _MI_Tool_ErrorList.Click += new EventHandler( _MI_Tool_ErrorList_Click );
#       endif
        //
        // _MI_Window
        //
        _MI_Window.Index = 4;
        _MI_Window.Text = "&Window";
        _MI_Window.MenuItems.AddRange(
                new MenuItem[]{
                    _MI_Window_Next,
                    _MI_Window_Previous,
                    _MI_Window_List,
                    _MI_Window_Separator
                }
            );
        //
        // _MI_Window_Next
        //
        _MI_Window_Next.Index = 0;
        _MI_Window_Next.Text = "&Next window¥tCtrl+Tab";
        _MI_Window_Next.ShowShortcut = false;
        _MI_Window_Next.Shortcut = Shortcut.CtrlF6;
        _MI_Window_Next.Click += new EventHandler( _MI_Window_Next_Click );
        //
        // _MI_Window_Previous
        //
        _MI_Window_Previous.Index = 1;
        _MI_Window_Previous.Text = "&Previous window¥tCtrl+Shift+Tab";
        _MI_Window_Previous.ShowShortcut = false;
        _MI_Window_Previous.Shortcut = Shortcut.CtrlShiftF6;
        _MI_Window_Previous.Click += new EventHandler( _MI_Window_Previous_Click );
        //
        // _MI_Window_List
        //
        _MI_Window_List.Index = 2;
        _MI_Window_List.Text = "List up &windows...";
```

```
            _MI_Window_List.Shortcut = Shortcut.F6;
            _MI_Window_List.Click += new EventHandler( _MI_Window_List_Click );
            //
            // _MI_Window_Separator
            //
            _MI_Window_Separator.Index = 3;
            _MI_Window_Separator.Text = "-";
            //
            // _MI_Braille
            //
            _MI_Braille.Index = 5;
            _MI_Braille.MenuItems.AddRange(
                    new MenuItem[] {_MI_Braille_Use, _MI_Braille_BrailleConfig,
_MI_Braille_DeviceConfig}
                );
            _MI_Braille.Text = "&Braille display";
            //
            // _MI_Braille_Use
            //
            _MI_Braille_Use.Index = 0;
            _MI_Braille_Use.Text = "&Use braille display";
            _MI_Braille_Use.Click += new EventHandler(_MI_Braille_Use_Click);
            //
            // _MI_Braille_BrailleConfig
            //
            _MI_Braille_BrailleConfig.Index = 1;
            _MI_Braille_BrailleConfig.Text = "&Braille preference...";
            _MI_Braille_BrailleConfig.Shortcut = Shortcut.CtrlB;
            _MI_Braille_BrailleConfig.Click
                += new EventHandler(_MI_Braille_BrailleConfig_Click);
            //
            // _MI_Braille_DeviceConfig
            //
            _MI_Braille_DeviceConfig.Index = 2;
            _MI_Braille_DeviceConfig.Text = "&Device preference...";
            _MI_Braille_DeviceConfig.Click
                += new EventHandler(_MI_Braille_DeviceConfig_Click);
#if ACCEDIT_IDE
            //
            // _ErrorListPanel
            //
            _ErrorListPanel.Dock = DockStyle.Bottom;
            _ErrorListPanel.Controls.Add( _ErrorListLabel );
            _ErrorListPanel.Controls.Add( _ErrorListView );
            _ErrorListPanel.Height = 121;
            //
            // _ErrorListLabel
            //
            _ErrorListLabel.Font = SystemInformation.MenuFont;
            _ErrorListLabel.Text = "Error List";
```

```
            _ErrorListLabel.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;
            _ErrorListLabel.Anchor = AnchorStyles.Left | AnchorStyles.Top |
AnchorStyles.Right;
            _ErrorListLabel.TabIndex = 0;
            _ErrorListLabel.Width = this.Width;
            _ErrorListLabel.Height = 21;
            //
            // _ErrorListView
            //
            ColumnHeader ch_message = new ColumnHeader();
            ch_message.Text = "Message";
            ch_message.Width = 480;
            ColumnHeader ch_file = new ColumnHeader();
            ch_file.Text = "File";
            ch_file.Width = 80;
            ColumnHeader ch_line = new ColumnHeader();
            ch_line.Text = "LineNumber";
            ch_line.Width = 40;
            _ErrorListView.Anchor = AnchorStyles.Left | AnchorStyles.Bottom |
AnchorStyles.Right;
            _ErrorListView.Width = this.Width;
            _ErrorListView.Height = 100;
            _ErrorListView.Top = 21;
            _ErrorListView.TabStop = false;
            _ErrorListView.TabIndex = 1;
            _ErrorListView.View = View.Details;
            _ErrorListView.GridLines = true;
            _ErrorListView.FullRowSelect = true;
            _ErrorListView.MultiSelect = false;
            _ErrorListView.Font = SystemInformation.MenuFont;
            _ErrorListView.Items.Add( _DummyListViewItem );
            _ErrorListView.Columns.AddRange(
                    new ColumnHeader[]{ ch_message, ch_file, ch_line }
                );
            _ErrorListView.KeyDown += new KeyEventHandler( _ErrorListView_KeyDown );
            _ErrorListView.ItemActivate += new EventHandler( _ErrorListView_ItemActivate
);
#endif
            //
            // _TabControl
            //
            _TabControl.Dock = DockStyle.Top;
            _TabControl.Height = 18;
            //
            // _TextBoxPanel
            //
            _TextBoxPanel.Dock = DockStyle.Fill;
            //
            // AiBEditForm
            //
```

```
            Controls.Add( _TextBoxPanel );
            Controls.Add( _TabControl );
#if ACCEDIT_IDE
            Controls.Add( _ErrorListPanel );
#endif
            AllowDrop = true;
            Size = new System.Drawing.Size( 640, 480 );
            Menu = _MainMenu;
            Name = "AiBEditForm";
            Text = "AiB Edit";
#           if !DEBUG
            WindowState = FormWindowState.Maximized;
#           endif
            DragDrop += new DragEventHandler( Form_DragDrop );
            DragEnter += new DragEventHandler( Form_DragEnter );
            ResumeLayout();
        }
        #endregion

        #region UI components
        MainMenu    _MainMenu    = new MainMenu();
        MenuItem    _MI_File             = new MenuItem();
        MenuItem    _MI_File_New         = new MenuItem();
        MenuItem    _MI_File_Open        = new MenuItem();
        MenuItem    _MI_File_Save        = new MenuItem();
        MenuItem    _MI_File_SaveAs      = new MenuItem();
        MenuItem    _MI_File_Separator   = new MenuItem();
        MenuItem    _MI_File_Reload      = new MenuItem();
        MenuItem    _MI_File_Reload_AnalyzeAgain    = new MenuItem();
        MenuItem    _MI_File_Reload_Separator    = new MenuItem();
        MenuItem    _MI_File_Reload_Sjis         = new MenuItem();
        MenuItem    _MI_File_Reload_Jis          = new MenuItem();
        MenuItem    _MI_File_Reload_Euc          = new MenuItem();
        MenuItem    _MI_File_Reload_Utf8         = new MenuItem();
        MenuItem    _MI_File_Separator2 = new MenuItem();
        MenuItem    _MI_File_Mru         = new MenuItem();
        MenuItem    _MI_File_Separator3 = new MenuItem();
        MenuItem    _MI_File_Close       = new MenuItem();
        MenuItem    _MI_File_Exit        = new MenuItem();
        MenuItem    _MI_Edit                 = new MenuItem();
        MenuItem    _MI_Edit_Undo            = new MenuItem();
        MenuItem    _MI_Edit_Separator1      = new MenuItem();
        MenuItem    _MI_Edit_Cut             = new MenuItem();
        MenuItem    _MI_Edit_Copy            = new MenuItem();
        MenuItem    _MI_Edit_Paste           = new MenuItem();
        MenuItem    _MI_Edit_Separator2      = new MenuItem();
        MenuItem    _MI_Edit_Find            = new MenuItem();
        MenuItem    _MI_Edit_FindNext        = new MenuItem();
        MenuItem    _MI_Edit_FindPrevious    = new MenuItem();
        MenuItem    _MI_Edit_Separator3      = new MenuItem();
```

```
MenuItem    _MI_Edit_SelectAll      = new MenuItem();
MenuItem    _MI_Edit_JumpToLine     = new MenuItem();
MenuItem    _MI_Edit_MatchedBracket = new MenuItem();
MenuItem    _MI_Speech              = new MenuItem();
MenuItem    _MI_Speech_CurrentLine  = new MenuItem();
MenuItem    _MI_Tool            = new MenuItem();
MenuItem    _MI_Tool_Mode       = new MenuItem();
MenuItem    _MI_Tool_Separator  = new MenuItem();
MenuItem    _MI_Tool_Outline    = new MenuItem();
MenuItem    _MI_Tool_Compile    = new MenuItem();
MenuItem    _MI_Tool_Build      = new MenuItem();
MenuItem    _MI_Tool_ErrorList  = new MenuItem();
MenuItem    _MI_Window          = new MenuItem();
MenuItem    _MI_Window_Next     = new MenuItem();
MenuItem    _MI_Window_Previous = new MenuItem();
MenuItem    _MI_Window_List     = new MenuItem();
MenuItem    _MI_Window_Separator= new MenuItem();
MenuItem    _MI_Braille                 = new MenuItem();
MenuItem    _MI_Braille_Use             = new MenuItem();
MenuItem    _MI_Braille_DeviceConfig    = new MenuItem();
MenuItem    _MI_Braille_BrailleConfig   = new MenuItem();
AiBTabControl   _TabControl = new AiBTabControl();
Panel       _TextBoxPanel   = new Panel();
Panel       _ErrorListPanel = new Panel();
Label       _ErrorListLabel = new Label();
ListView    _ErrorListView  = new ListView();
ListViewItem    _DummyListViewItem = new ListViewItem(
        new String[]{"(no error)", "", "", ""}
    );
#endregion // UI components

#region Utility
class Utl
{
    /// <summary>
    /// 指定した半角文字単位の桁位置が、
    /// 指定文字列の何文字目に対応するのかを計算する。
    /// </summary>
    /// <param name="str">指定文字列</param>
    /// <param name="halfWidthIndex">半角文字単位の桁位置インデックス</param>
    /// <param name="tabWidth">タブ幅</param>
    /// <returns>対応する文字インデックス</returns>
    /// <remarks>
    /// もし指定した桁位置が非半角文字の中ならば、その文字のインデックスを返す。
    /// </remarks>
    public static int
    CalcCharIndexFromHalfWidthIndex( string str, int halfWidthIndex, int tabWidth
)
    {
        int charPos;
```

```
    int hwPos           = 0;
    int offsetToNext    = 0;

    // 先頭から一文字ずつ見ていき、半角文字単位での幅を積算していく
    charPos = 0;
    while( charPos < str.Length )
    {
        char ch = str[charPos];

        // タブ？
        if( '¥t' == ch )
        {
            offsetToNext = tabWidth - (hwPos % tabWidth);
        }
        // 半角文字？
        else if( IsHalfWidthChar(ch) )
        {
            offsetToNext = 1;
        }
        // 全角文字？
        else
        {
            offsetToNext = 2;
        }

        // 次の文字へ
        hwPos += offsetToNext;
        charPos++;

        // 指定された幅を超えたなら、超える直前の文字カウントを返す
        if( halfWidthIndex < hwPos )
        {
            return charPos - 1;
        }
    }

    // 全文字を見終わったのに、まだ指定された幅に達しない。
    // つまり指定された幅が大きすぎるという事なので、行末とする
    return str.Length;
}

/// <summary>
/// 指定文字が半角文字かどうか判定
/// </summary>
/// <param name="ch">診断対象文字</param>
/// <returns>半角だったら true</returns>
static bool IsHalfWidthChar( char ch )
{
    // is in "Basic-Latin"?
    if( 0x20 <= ch && ch < 0x7F )
```

```
                {
                    return true;
                }
                // is in "Half-width CJK punctuation"?
                if( 0xFF61 < ch && ch < 0xFF64 )
                {
                    return true;
                }
                // is in "Half-width Katakana variants"?
                if( 0xFF65 < ch && ch < 0xFF9F )
                {
                    return true;
                }
                // is in "Half-width Hangle variants"?
                if( 0xFFA0 < ch && ch < 0xFFBD )
                {
                    return true;
                }
                // is in "Half-width Symbol variants"?
                if( 0xFFE8 < ch && ch < 0xFFEE )
                {
                    return true;
                }

                return false;
            }
        }
        #endregion
    }
}
```

CppOutliner.cs

```
// file   : CppOutliner.cs
// brief  : Outline parser for C family (C, C++, C#, Java)
// update : 2007-02-25
//==========================================================
using System;
using System.Text;
using System.Windows.Forms;

namespace Sgry.AiBTools.AiBEdit
{
    /// <summary>
    /// C 系言語用のアウトライン解析器
    /// </summary>
    internal class CppOutliner : Outliner
    {
        #region Constants
        static readonly char[] Brackets = new char[] { '{', '}' };
```

```
        static readonly char[] LogicalLineEndChar = new char[] { '{', '}', ';' };
        #endregion

        /// <summary>
        /// インスタンスを生成
        /// </summary>
        /// <param name="dialog">シンボルを表示するアウトラインダイアログ</param>
        /// <param name="source">解析する文書</param>
        /// <param name="caretIndex">現在のキャレット位置</param>
        public CppOutliner( OutlineDialog dialog, string source, int caretIndex )
            : base( dialog, source, caretIndex )
        {}

        /// <summary>
        /// 文書からシンボルを抽出してツリーコントロールに表示
        /// </summary>
        public override void Parse()
        {
            // parse source code
            try
            {
                Parse( _Dialog.TreeView, Strip(_Source), _CaretIndex );
            }
            catch( Exception ex )
            {
                Console.Error.Write( ex.StackTrace );
            }

            // if no identifier found, display message
            if( _Dialog.TreeView.Nodes.Count == 0 )
            {
                Sgry.Localizer localizer = new Sgry.Localizer();
                string  dummyNodeLabel;
                TreeNode dummyNode;

                // get localized label
                localizer.LoadResourceFile();
                dummyNodeLabel = localizer.TryGetString( "Msg_NoSymbolsFound", "(No
symbols found)" );

                // add dummy node
                dummyNode = new TreeNode( dummyNodeLabel );
                _Dialog.TreeView.Nodes.Add( dummyNode );
                dummyNode.Tag = _CaretIndex;
            }
        }

        #region 解析ロジック
        static void Parse( TreeView treeView, string source, int caretIndex )
        {
```

```
            int         pos = 0;
            TreeNode    currentParent = null;
            TreeNode    initSelNode = null;

            // parse
            pos = FindNextExpectedPosition( source, pos );
            while( pos != -1 )
            {
                if( source[pos] == '{' )
                {
                    Parse_OnOpenBracket( treeView, ref initSelNode, source, caretIndex,
ref pos, ref currentParent );
                }
                else if( source[pos] == '}' )
                {
                    if( currentParent != null )
                        currentParent = currentParent.Parent;
                }
                if( pos == -1 )
                {
                    break; // some error occured in inner process
                }

                // find next
                pos = FindNextExpectedPosition( source, pos+1 );
            }

            // select initial selection node
            if( initSelNode != null )
            {
                Sgry.Win32.TreeView_SelectItem( treeView.Handle, initSelNode.Handle );
            }
        }

        /// <summary>
        /// Find next position where a symbol was expected to exist around.
        /// </summary>
        static int FindNextExpectedPosition( string source, int startPos )
        {
            int foundPos = startPos;

            // find keyword near the next identifier
            foundPos = source.IndexOfAny( Brackets, foundPos );
            while( foundPos != -1 && foundPos < source.Length )
            {
                if( source[foundPos] == '{'
                    || source[foundPos] == '}' )
                {
                    return foundPos; // found
                }
```

```
            // find keyword near the next identifier
            foundPos = source.IndexOfAny( Brackets, foundPos+1 );
        }

        return -1; // no more IDs
    }

    static void Parse_OnOpenBracket( TreeView treeView, ref TreeNode initSelNode,
string source, int caretIndex, ref int pos, ref TreeNode currentParent )
    {
        bool    succeeded;

        // class?
        succeeded = TryParseContainerSymbol( treeView, ref currentParent, ref
initSelNode, source, caretIndex, ref pos, "class" );
        if( succeeded )  return;

        // interface?
        succeeded = TryParseContainerSymbol( treeView, ref currentParent, ref
initSelNode, source, caretIndex, ref pos, "interface" );
        if( succeeded )  return;

        // namespace?
        succeeded = TryParseContainerSymbol( treeView, ref currentParent, ref
initSelNode, source, caretIndex, ref pos, "namespace" );
        if( succeeded )  return;

        // struct?
        succeeded = TryParseContainerSymbol( treeView, ref currentParent, ref
initSelNode, source, caretIndex, ref pos, "struct" );
        if( succeeded )  return;

        // enum?
        succeeded = TryParseContainerSymbol( treeView, ref currentParent, ref
initSelNode, source, caretIndex, ref pos, "enum" );
        if( succeeded )  return;

        // union?
        succeeded = TryParseContainerSymbol( treeView, ref currentParent, ref
initSelNode, source, caretIndex, ref pos, "union" );
        if( succeeded )  return;

        // function / method?
        succeeded = TryParseFunctionSymbol( treeView, ref currentParent, ref
initSelNode, source, caretIndex, ref pos );
        if( succeeded )  return;
    }

    static bool TryParseContainerSymbol( TreeView treeView, ref TreeNode
```

```
currentParent, ref TreeNode initSelNode,
                string source, int caretIndex,
                ref int pos, string symbolType )
        {
            int idKeyPos, idPos;
            string idName;

            idKeyPos = FindWordInLogicalLine( source, pos-1, symbolType );
            if( idKeyPos == -1 )
            {
                return false;
            }

            // extract name and add node
            idName = ExtractSymbolNameFromLine( source.Substring(idKeyPos,
pos-idKeyPos), symbolType );
            if( idName == null )
            {
                return false;
            }
            idPos = Utl.NextWordPos( source, idKeyPos );

            // insert tree node
            TreeNode node = new TreeNode( /*symbolType[0] + " " + */idName );
            node.Tag = idPos;
            if( currentParent == null )
            {
                treeView.Nodes.Add( node );
            }
            else
            {
                currentParent.Nodes.Add( node );
            }

            // このノードを初期選択するか判定
            if( Utl.CurrentLineHeadPos(source, idPos) <= caretIndex )
                initSelNode = node;

            // このノードをカレントノードに
            currentParent = node;

            return true;
        }

        static bool TryParseFunctionSymbol( TreeView treeView, ref TreeNode
currentParent, ref TreeNode initSelNode,
                string source, int caretIndex, ref int pos )
        {
            int     idKeyPos, idPos, idNameEndPos, idNameBeginPos;
            string  idName;
```

```
        TreeNode node;
        int     scopeOpPos;

        idKeyPos = FindCharInLogicalLine( source, pos-1, '(' );
        if( idKeyPos == -1 )
        {
            return false;
        }

        // extract function name
        idNameEndPos = Utl.PrevNonWhiteSpacePos( source, idKeyPos-1 ) + 1;
        idNameBeginPos = Utl.PrevWhiteSpacePos( source, idNameEndPos-1 ) + 1;
        if( idNameBeginPos <= 0 || idNameBeginPos <= 0 )
        {
            return false;
        }
        idName = source.Substring( idNameBeginPos, idNameEndPos - idNameBeginPos );
        idPos = idNameEndPos - idName.Length + 1;

        // is this a member function?
        scopeOpPos = idName.IndexOf( "::" );
        if( scopeOpPos != -1 )
        {
            // --- add node as a child of proper class ---
            string className = idName.Substring( 0, scopeOpPos );

            // find the class
            TreeNode classNode = Utl.FindNodeByNameIn( treeView.Nodes, /*"c " +
*/className );
            if( classNode == null )
            {
                // the class was not added yet. then add.
                classNode = new TreeNode( /*"c " + */className );
                if( currentParent == null )
                    treeView.Nodes.Add( classNode );
                else
                    currentParent.Nodes.Add( classNode );
            }

            // add node
            node = new TreeNode( /*"m " + */idName.Substring(scopeOpPos+2) ); //
2=="::".Length
            node.Tag = idPos;
            classNode.Nodes.Add( node );
        }
        else
        {
            // --- add node to current node ---
            node = new TreeNode();
            node.Tag = idPos;
```

```
            if( currentParent == null )
            {
                node.Text = /*"f " + */idName;
                treeView.Nodes.Add( node );
            }
            else
            {
                // if parent is class or interface, this is a method
                /*string prefix = "f ";

                char pType = currentParent.Text[0];
                if( pType == 'c' || pType == 'i' ) // class or interface
                    prefix = "m ";*/

                node.Text = /*prefix + */idName;
                currentParent.Nodes.Add( node );
            }
        }

        // メソッドの内容をスキップ
        pos = FindPairedCloseBracket( source, pos );

        // このノードを初期選択するか判定
        if( Utl.CurrentLineHeadPos(source, idPos) <= caretIndex )
            initSelNode = node;

        return true;
    }

    /// <summary>
    /// 指定位置が含まれる論理行の行頭から単語を検索。
    /// </summary>
    static int FindWordInLogicalLine( string source, int startPos, string word )
    {
        int logicalLineHeadPos;
        int wordBeginPos;
        string foundWord;

        logicalLineHeadPos = source.LastIndexOfAny( LogicalLineEndChar, startPos ) +
1;
        //考慮不要 if( logicalLineHeadPos == 0 )

        // get the word position
        wordBeginPos = source.IndexOf( word, logicalLineHeadPos, startPos -
logicalLineHeadPos );
        if( wordBeginPos <= 0 || source.Length <= wordBeginPos )
            return -1; // no such word

        foundWord = Utl.WordAt( source, wordBeginPos );
        if( foundWord != word )
```

```
            return -1;

        return wordBeginPos;
    }

    /// <summary>
    /// 指定位置が含まれる論理行の行頭から文字を検索。
    /// </summary>
    static int FindCharInLogicalLine( string source, int startPos, char ch )
    {
        int logicalLineHeadPos;
        int wordBeginPos;

        logicalLineHeadPos = source.LastIndexOfAny( LogicalLineEndChar, startPos ) +
1;
        //考慮不要 if( logicalLineHeadPos == 0 )

        // get the word position
        wordBeginPos = source.IndexOf( ch, logicalLineHeadPos, startPos -
logicalLineHeadPos );
        if( wordBeginPos <= 0 || source.Length <= wordBeginPos )
            return -1; // no such word

        return wordBeginPos;
    }

    /// <summary>
    /// ソース中の一行からシンボル名を抽出。
    /// 指定キーワード直後の単語を検索して返す。
    /// </summary>
    /// <returns>失敗するとnull</returns>
    static string ExtractSymbolNameFromLine( string line, string key )
    {
        int pos;

        pos = line.IndexOf( key );
        if( pos == -1 )
        {
            return null;
        }

        int classNamePos = Utl.NextWordPos( line, pos );
        if( classNamePos == -1 )
        {
            return null;
        }

        return Utl.WordAt( line, classNamePos );
    }
```

```
/// <summary>
/// 指定位置にある開き中括弧の対となる閉じ中括弧を検索
/// </summary>
static int FindPairedCloseBracket( string source, int openBracketPos )
{
    int depth = 1;
    int pos = openBracketPos;

    try
    {
        // 開き中括弧の位置から順に閉じ中括弧を探す
        while( 0 < depth && 0 < pos && pos + 1 < source.Length )
        {
            // 次の中括弧を探す
            pos = source.IndexOfAny( Brackets, pos + 1 );
            if( source[pos] == '{' )
            {
                depth++;
            }
            else
            {
                depth--;
            }
        }

        return pos;
    }
    catch( IndexOutOfRangeException )
    {
        return -1;
    }
}
#endregion

#region コメント等の除去ロジック
/// <summary>
/// アウトライン解析をする上で邪魔になるコメント等を取り除く
/// </summary>
/// <param name="src">コメント等を取り除きたいソース</param>
/// <returns>コメント等が取り除かれたソース</returns>
public static string Strip( string src )
{
    StringBuilder buffer = new StringBuilder( src.Length );

    for( int i=0; i<src.Length; i++ )
    {
        // string literal begin?
        if( src[i] == '¥'' )
        {
            // string literal found. replace into whitespace
```

```
                i = Utl.StripToPatternFrom( src, i, "¥'", ref buffer );
            }
            // charcter literal begin?
            else if( src[i] == '¥"' )
            {
                i = Utl.StripToPatternFrom( src, i, "¥"", ref buffer );
            }
            // comment begin?
            else if( src[i] == '/' )
            {
                if( src[i+1] == '/' )
                {
                    // one-line comment found
                    int nextLineHeadPos = Utl.NextLineHeadPos( src, i+1 );
                    if( nextLineHeadPos == -1 )
                    {
                        break; // fatal error
                    }
                    for( int j=0; j<nextLineHeadPos - i - 1; j++ )
                        buffer.Append( ' ' );
                    i = nextLineHeadPos - 1; // -1 for i++ at next loop begin
                }
                else if( src[i+1] == '*' )
                {
                    // block comment found
                    i = Utl.StripToPatternFrom( src, i, "*/", ref buffer );
                }
            }

            // add this char to comment-stripped buffer
            buffer.Append( src[i] );
        }

        return buffer.ToString();
    }
    #endregion
  }
}
```

RubyOutliner.cs

```
// file   : RubyOutliner.cs
// brief  : Outline parser for Ruby
// update : 2007-01-22
//=========================================================
using System;
using System.Text;
using System.Text.RegularExpressions;
using System.Collections;
using System.Windows.Forms;
```

```
namespace Sgry.AiBTools.AiBEdit
{
    /// <summary>
    /// Ruby 用アウトライン解析器
    /// </summary>
    internal class RubyOutliner : Outliner
    {
        /// <summary>
        /// インスタンスを生成
        /// </summary>
        /// <param name="dialog">シンボルを表示するアウトラインダイアログ</param>
        /// <param name="source">解析する文書</param>
        /// <param name="caretIndex">現在のキャレット位置</param>
        public RubyOutliner( OutlineDialog dialog, string source, int caretIndex )
            : base( dialog, source, caretIndex )
        {}

        /// <summary>
        /// 文書からシンボルを抽出してツリーコントロールに表示
        /// </summary>
        public override void Parse()
        {
            // parse source code
            try
            {
                Parse( _Dialog.TreeView, Strip(_Source), _CaretIndex );
            }
            catch( Exception ex )
            {
                Console.Error.Write( ex.StackTrace );
            }

            // if no identifier found, display message
            if( _Dialog.TreeView.Nodes.Count == 0 )
            {
                Sgry.Localizer localizer = new Sgry.Localizer();
                string  dummyNodeLabel;
                TreeNode dummyNode;

                // get localized label
                localizer.LoadResourceFile();
                dummyNodeLabel = localizer.TryGetString( "Msg_NoSymbolsFound", "(No
symbols found)" );

                // add dummy node
                dummyNode = new TreeNode( dummyNodeLabel );
                _Dialog.TreeView.Nodes.Add( dummyNode );
                dummyNode.Tag = _CaretIndex;
            }
        }
```

```
#region 解析ロジック
static void Parse( TreeView treeView, string source, int caretIndex )
{
    Stack       endOwnerIsContainer = new Stack();
    TreeNode    currentParent = null, initSelNode = null;
    int         pos;
    string      token;

    // read each tokens and if it is a symbol to be displayed in the outline, add it
to the tree
    pos = 0;
    token = Utl.WordAt( source, pos );
    while( token != null )
    {
        //--- analyze this token ---
        if( token == "module" )
        {
            AddContainerAt( treeView, ref currentParent, ref initSelNode,
caretIndex, source, Utl.NextWordPos(source, pos) );
            endOwnerIsContainer.Push( true );
        }
        else if( token == "class" )
        {
            AddContainerAt( treeView, ref currentParent, ref initSelNode,
caretIndex, source, Utl.NextWordPos(source, pos) );
            endOwnerIsContainer.Push( true );
        }
        else if( token == "def" )
        {
            AddNonContainerAt( treeView, ref currentParent, ref initSelNode,
caretIndex, source, Utl.NextWordPos(source, pos) );
            endOwnerIsContainer.Push( false );
        }
        else if( token == "if" || token == "while" || token == "unless" )
        {
            if( IsModifierIfWhile(source, pos) == false )
                endOwnerIsContainer.Push( false );
        }
        else if( token == "for" || token == "do" || token == "begin" || token ==
"case" || token == "until" )
        {
            endOwnerIsContainer.Push( false );
        }
        else if( token == "end" )
        {
            bool ownerIsContainer = (bool)endOwnerIsContainer.Pop();
            if( ownerIsContainer )
            {
                currentParent = currentParent.Parent;
```

```
                }
            }

            // goto next token
            token = Utl.NextNonWhiteSpaceToken( source, ref pos );
        }

        // select initial selection node
        if( initSelNode != null )
        {
            Sgry.Win32.TreeView_SelectItem( treeView.Handle, initSelNode.Handle );
        }
    }

    /// <summary>
    /// if/unless/while が修飾子かどうかを判定
    /// </summary>
    static bool IsModifierIfWhile( string source, int pos )
    {
        if( pos <= 0 )
        {
            return false; // if/while より前に文字がないので if/while 修飾子ではない
        }

        // get previous non white space char
        int prevChPos = Utl.PrevNonWhiteSpacePos( source, pos - 1 );
        int lineHeadPos = Utl.CurrentLineHeadPos( source, pos );

        if( prevChPos < lineHeadPos )
        {
            return false; // 行頭の if/while; つまり普通の if/while。
        }
        else if( source[prevChPos] == '='
            || (source[prevChPos-1] == '=' && source[prevChPos] == '>') )
        {
            return false; // 代入式の右辺に if がきた。これは C の三項演算子と同じ意味の
if/else/end。
        }

        return true;
    }

    static void AddContainerAt( TreeView treeView, ref TreeNode currentNode,
            ref TreeNode initSelNode, int caretIndex,
            string source, int index )
    {
        TreeNode node = new TreeNode();

        // make node
        node.Text = Utl.WordAt( source, index );
```

```
            node.Tag = index;

            // but wait, it can be a singleton class. if so, add the object name
            if( node.Text == "<<" )
            {
                int nextWordPos = Util.NextWordPos( source, index );
                if( nextWordPos != -1 )
                {
                    node.Text += " " + Util.WordAt( source, nextWordPos );
                }
            }

            // add node
            if( currentNode == null )
            {
                treeView.Nodes.Add( node );
            }
            else
            {
                currentNode.Nodes.Add( node );
            }

            // if seek position didnt go over where caret is, keep this node as candidate for
initial selection
            if( Util.CurrentLineHeadPos(source, index) <= caretIndex )
            {
                initSelNode = node;
            }

            currentNode = node;
        }

        static void AddNonContainerAt( TreeView treeView, ref TreeNode currentNode,
                ref TreeNode initSelNode, int caretIndex,
                string source, int index )
        {
            TreeNode node = new TreeNode();

            // make node
            node.Text = GetMethodNameAt( source, index );
            node.Tag = index;

            // add node
            if( currentNode == null )
            {
                treeView.Nodes.Add( node );
            }
            else
            {
                currentNode.Nodes.Add( node );
```

```
            }

            // if seek position didnt go over where caret is, keep this node as candidate for
initial selection
            if( Utl.CurrentLineHeadPos(source, index) <= caretIndex )
            {
                initSelNode = node;
            }
        }

        static string GetMethodNameAt( string source, int index )
        {
            int endPos;

            // try extract token between here and where first '(', ';' or whitespaces is
appeared
            for( endPos=index; endPos<source.Length; endPos++ )
            {
                char ch = source[endPos];
                if( Char.IsWhiteSpace(ch)
                    || ch == '('
                    || ch == ';' )
                {
                    return source.Substring( index, endPos - index );
                }
            }

            // the token just after the 'def' is the last token. may not be valid syntax but
try parse.
            return source.Substring( index );
        }
        #endregion

        #region コメント等の除去ロジック
        /// <summary>
        /// Ruby ソースからコメント等を取り除く
        /// </summary>
        /// <param name="src">コメント等を取り除きたいソース</param>
        /// <returns>コメント等が取り除かれたソース</returns>
        public static string Strip( string src )
        {
            StringBuilder buffer = new StringBuilder( src.Length );

            for( int i=0; i<src.Length; i++ )
            {
                //--- strip here document ---
                if( src[i] == '<'
                    && i+1 < src.Length && src[i+1] == '<'
                    && i+2 < src.Length && Char.IsWhiteSpace(src[i+2]) == false )
                {
```

```
                // possibly here document. check
                int hdEndPos = GetHereDocumentEndAt( src, i );
                if( hdEndPos != -1 )
                {
                    i = Utl.StripToPatternFrom( src, i, hdEndPos, ref buffer );
                }
            }
            //--- strip string literals ---
            // single quote string literal begin?
            else if( src[i] == '¥'' )
            {
                i = Utl.StripToPatternFrom( src, i, "¥'", ref buffer );
            }
            // string literal begin?
            else if( src[i] == '¥"' )
            {
                i = Utl.StripToPatternFrom( src, i, "¥"", ref buffer );
            }
            // command output begin?
            else if( src[i] == '`' )
            {
                i = Utl.StripToPatternFrom( src, i, "`", ref buffer );
            }
            // % expression begin?
            else if( src[i] == '%' )
            {
                char p_next = src[i+1];
                if( !Char.IsLetterOrDigit(p_next) )
                {
                    i = Utl.StripToPatternFrom( src, i, p_next.ToString(), ref buffer
);
                }
                else
                {
                    if( p_next == 'q' || p_next == 'Q' || p_next == 'w' || p_next == 'W'
                        || p_next == 'x' || p_next == 'r' || p_next == 's' )
                        i = Utl.StripToPatternFrom( src, i, src[i+2].ToString(), ref
buffer );
                }
            }
            //--- comments ---
            // one-line-comment begin?
            else if( src[i] == '#' )
            {
                // get next line head pos
                int nextLineHeadPos = Utl.NextLineHeadPos( src, i+1 );
                if( nextLineHeadPos == -1 )
                {
                    break; // final line not terminated by EOL
                }
```

```
            // replace between here to there with whitespaces
            for( int j=0; j<nextLineHeadPos - i - 1; j++ )
            {
                buffer.Append( ' ' );
            }
            i = nextLineHeadPos - 1;
        }

        // add this char to comment-stripped buffer
        buffer.Append( src[i] );
    }

    return buffer.ToString();
}

static int GetHereDocumentEndAt( string src, int index )
{
    bool isIndentHd = false;
    Regex hdIdPattern, hdEndPattern;
    Match hdIdMatchRes, hdEndMatchRes;
    string hdId;
    int hdEndIdPos;
    int hdEndLineHeadPos;

    if( src.Length < index+2 )
    {
        return -1; // too short. never be here document.
    }

    // validate here doc begin pattern
    hdIdPattern = new Regex( @"<<(-?)([^¥r|^¥n]+)" );
    hdIdMatchRes = hdIdPattern.Match( src, index );
    if( hdIdMatchRes.Success != true )
    {
        return -1;
    }

    // check if the here doc terminator would be indented
    if( hdIdMatchRes.Groups[1].ToString() != String.Empty )
    {
        isIndentHd = true;
    }

    // find terminator of the here doc
    hdId = hdIdMatchRes.Groups[2].ToString();
    hdEndIdPos = src.IndexOf( hdId, index + 2 + hdId.Length ); // 2=="<<".Length
    if( hdEndIdPos == -1 )
    {
        return -1;
```

```
            }

            // validate the terminator
            hdEndLineHeadPos = Utl.CurrentLineHeadPos( src, hdEndIdPos );
            if( isIndentHd )
                hdEndPattern = new Regex( @"¥s*" + hdId + @"[¥r|¥n]" );
            else
                hdEndPattern = new Regex( hdId + @"[¥r|¥n]" );
            hdEndMatchRes = hdEndPattern.Match( src, hdEndLineHeadPos );
            if( hdEndMatchRes.Success != true )
            {
                return -1;
            }

            return hdEndIdPos + hdId.Length;
        }
        #endregion
    }
}
```

AutoSpeaker.cs

```
// file    : AutoSpeaker.cs
// brief   : Class to make screen readers speak
// author  : SGRY (YAMAMOTO Suguru)
// update  : 2007-02-25 (SGRY)
// version : 1.2.1
// license : MIT License (see END of this file)
//=========================================================
using System;
using System.Text;
using Microsoft.Win32;
using System.Runtime.InteropServices;

namespace Sgry
{
    /// <summary>
    /// 起動しているスクリーンリーダーに発話させるためのインタフェース
    /// </summary>
    public interface ISpeaker
    {
        /// <summary>
        /// 文字列を読み上げさせる
        /// </summary>
        /// <param name="message">読み上げさせる文字列</param>
        void Speak( string message );

        /// <summary>
        /// 読み上げを停止
        /// </summary>
        void Stop();
```

```
    /// <summary>
    /// 読み上げに「間」を挿入する
    /// </summary>
    void Wait();
}

/// <summary>
/// 起動しているスクリーンリーダーを自動的に選択して読み上げを実行させるクラス
/// </summary>
public class AutoSpeaker
{
    AutoSpeaker()
    {}

    /// <summary>
    /// 起動中のスクリーンリーダーを制御するスピーカオブジェクトを取得
    /// </summary>
    /// <returns>起動中スクリーンリーダ用のスピーカオブジェクト</returns>
    public static ISpeaker Instance
    {
        get
        {
            if( JawsSpeaker.IsAlive() )
                return JawsSpeaker.Instance;
            else if( PcTalkerSpeaker.IsAlive() )
                return PcTalkerSpeaker.Instance;
            else if( FocusTalkSpeaker.IsAlive() )
                return FocusTalkSpeaker.Instance;
            else if( XpReaderSpeaker.IsAlive() )
                return XpReaderSpeaker.Instance;
            else
                return DummySpeaker.Instance;
        }
    }

    /// <summary>
    /// 起動中のスクリーンリーダーを制御するスピーカオブジェクトを取得できるか判定
    /// </summary>
    /// <returns>取得できる場合は true</returns>
    public static bool IsAlive()
    {
        if( JawsSpeaker.IsAlive() )
        {
            return true;
        }
        else if( PcTalkerSpeaker.IsAlive() )
        {
            return true;
        }
```

```
        else if( FocusTalkSpeaker.IsAlive() )
        {
            return true;
        }
        else if( XpReaderSpeaker.IsAlive() )
        {
            return true;
        }
        return false;
    }
}

/// <summary>
/// 何も発話しない、ダミーのスピーカークラス
/// </summary>
public class DummySpeaker : ISpeaker
{
    DummySpeaker(){}
    public void Speak( string message ){}
    public void Stop(){}
    public void Wait(){}

    static ISpeaker _Instance = null;
    public static ISpeaker Instance
    {
        get
        {
            if( _Instance == null )
                _Instance = new DummySpeaker();
            return _Instance;
        }
    }
}

/// <summary>
/// PC-Talker を単純なスピーカーとして使うためのクラス
/// </summary>
public class PcTalkerSpeaker : ISpeaker
{
    static PcTalkerSpeaker _Instance = null;

    PcTalkerSpeaker()
    {}

    /// <summary>
    /// 唯一のインスタンスを取得
    /// </summary>
    public static ISpeaker Instance
    {
        get
```

```
        // disable reading punctuation
        UInt32 readsKuten;
        readsKuten = PCTKGetStatus( PS_STATUS, new IntPtr(PKDI_KUTEN), IntPtr.Zero );
        PCTKSetStatus( PS_STATUS, new IntPtr(PKDI_KUTEN), IntPtr.Zero );

        // read punctuation to wait a moment
        PCTKPRead( "、", 5, false );

        // restore setting of reading punctuation
        PCTKSetStatus( PS_STATUS, new IntPtr(PKDI_KUTEN), new IntPtr(readsKuten) );
    }

    #region Inner functions
    [DllImport("PCTKUSR.dll")]
    static extern bool PCTKStatus();

    [DllImport("PCTKUSR.dll")]
    static extern UInt32 PCTKGetStatus( UInt32 item, IntPtr param1, IntPtr param2 );

    [DllImport("PCTKUSR.dll")]
    static extern UInt32 PCTKSetStatus( UInt32 item, IntPtr param1, IntPtr param2 );

    [DllImport("PCTKUSR.dll", CharSet=CharSet.Ansi)]
    static extern void PCTKPRead( string message, Int32 priority, bool analyze );

    [DllImport("PCTKUSR.dll")]
    static extern void PCTKVReset();
    #endregion
}

/// <summary>
/// JAWS を単純なスピーカーとして使うためのクラス
/// </summary>
public class JawsSpeaker : ISpeaker
{
    static ISpeaker _Instance = null;
    static string _JawsDir = null;

    JawsSpeaker()
    {}

    /// <summary>
    /// 唯一のインスタンスを取得
    /// </summary>
    public static ISpeaker Instance
    {
        get
        {
            if( _Instance == null )
            {
```

```
    {
        if( _Instance == null )
        {
            _Instance = new PcTalkerSpeaker();
        }
        return _Instance;
    }
}

/// <summary>
/// PC-Talker をスピーカーとして使えるか判定
/// </summary>
/// <returns>使える場合は true</returns>
public static bool IsAlive()
{
    try
    {
        return PCTKStatus();
    }
    catch( DllNotFoundException )
    {
        return false;
    }
}

/// <summary>
/// 文字列を読み上げさせる
/// </summary>
/// <param name="message">読み上げさせる文字列</param>
public void Speak( string message )
{
    PCTKPRead( message, 5, false );
}

/// <summary>
/// 読み上げを停止
/// </summary>
public void Stop()
{
    PCTKVReset();
}

/// <summary>
/// 読み上げに「間」を挿入する
/// </summary>
public void Wait()
{
    const int PS_STATUS = 0x00010001;
    const int PKDI_KUTEN = 17;
```

```
            _Instance = new JawsSpeaker();
        }
        return _Instance;
    }
}

/// <summary>
/// JAWS をスピーカーとして使えるか判定
/// </summary>
/// <returns>使える場合は true</returns>
public static bool IsAlive()
{
    string env_path;

    // get installation target of JAWS (for 6/7)
    GetJawsDir();
    if( _JawsDir == null )
    {
        return false; // not installed
    }

    // add to %PATH% env_var for DllImport (effects this process only)
    env_path = GetEnvVar( "PATH" );
    if( env_path.IndexOf(_JawsDir) == -1 )
    {
        SetEnvVar( "PATH", _JawsDir+";"+env_path );
    }

    // ensure JAWS is running
    try
    {
        if( JFWSayString("", false) == false )
        {
            return false; // not running
        }
    }
    catch( DllNotFoundException )
    {
        return false;
    }

    return true;
}

/// <summary>
/// 文字列を読み上げさせる
/// </summary>
/// <param name="message">読み上げさせる文字列</param>
public void Speak( string message )
{
```

```
    JFWSayString( message, true );
}

/// <summary>
/// 読み上げを停止
/// </summary>
public void Stop()
{
    //JFWStopSpeech(); // this API do nothing in Suguru's environment...
    JFWRunScript( "StopSpeech" );
}

/// <summary>
/// 読み上げに「間」を挿入する
/// </summary>
public void Wait()
{
    JFWSayString( "、", false );
}

#region Inner functions
static void GetJawsDir()
{
    RegistryKey key;

    if( _JawsDir == null )
    {
        // try to get directory of JAWS 7.1
        key = Registry.LocalMachine.OpenSubKey(
                @"SOFTWARE¥Freedom Scientific¥JAWS¥7.10"
            );
        if( key != null )
        {
            _JawsDir = (string)key.GetValue( "Target" );
        }

        // try to get directory of JAWS 6.2
        key = Registry.LocalMachine.OpenSubKey(
                @"SOFTWARE¥Freedom Scientific¥JAWS¥6.20"
            );
        if( key != null )
        {
            _JawsDir = (string)key.GetValue( "Target" );
        }
    }
}

static string GetEnvVar( string name )
{
    StringBuilder    str    = null;
```

```
    uint            length;
    uint            rc; // result code

    // get variable length
    length = GetEnvironmentVariable( name, str, (uint)0 );
    if( length == 0 )
    {
        return null; // no such variable
    }

    // get variable
    str = new StringBuilder( (int)length );
    rc = GetEnvironmentVariable( name, str, length );
    if( rc == 0 )
    {
        return null; // failed to get variable
    }

    return str.ToString();
}

static void SetEnvVar( string name, string value )
{
    bool rc; // result code

    rc = SetEnvironmentVariable( name, value );
    if( rc == false )
    {
        throw new Exception();
    }
}

[DllImport("jfwapi.dll")]
static extern bool JFWSayString( string message, bool interrupt );

//[DllImport("jfwapi.dll")]
//static extern bool JFWStopSpeech();

[DllImport("jfwapi.dll", CharSet=CharSet.Ansi)]
static extern bool JFWRunScript( string scriptName );

[DllImport("kernel32.dll",
    EntryPoint="SetEnvironmentVariableW", CharSet=CharSet.Unicode)]
static extern bool SetEnvironmentVariable( string name, string val );

[DllImport("kernel32.dll",
    EntryPoint="GetEnvironmentVariableW", CharSet=CharSet.Unicode)]
static extern uint
GetEnvironmentVariable( string name, StringBuilder str, uint strLength );
#endregion
```

```
}

/// <summary>
/// FocusTalk を単純なスピーカーとして使うためのクラス
/// </summary>
public class FocusTalkSpeaker : ISpeaker
{
    static ISpeaker _Instance = null;

    FocusTalkSpeaker()
    {}

    /// <summary>
    /// 唯一のインスタンスを取得
    /// </summary>
    public static ISpeaker Instance
    {
        get
        {
            if( _Instance == null )
            {
                _Instance = new FocusTalkSpeaker();
            }
            return _Instance;
        }
    }

    /// <summary>
    /// FocusTalk をスピーカーとして使えるか判定
    /// </summary>
    /// <returns>使える場合は true</returns>
    public static bool IsAlive()
    {
        // API DLL の関数を正常に P/Invoke できれば利用可能
        try
        {
            int rc = FT_OutputVoiceText( "", 0 );
            if( rc == 0 )
            {
                return false;
            }
        }
        catch( DllNotFoundException )
        {
            return false;
        }

        return true;
    }
```

```
    /// <summary>
    /// 文字列を読み上げさせる
    /// </summary>
    /// <param name="message">読み上げさせる文字列</param>
    public void Speak( string message )
    {
        FT_OutputVoiceText( message, 0 );
    }

    /// <summary>
    /// 読み上げを停止
    /// </summary>
    public void Stop()
    {
        FT_StopSound();
    }

    /// <summary>
    /// 読み上げに「間」を挿入する
    /// </summary>
    public void Wait()
    {
        System.Diagnostics.Debug.Fail( "Sgry.FocusTalkSpeaker.Wait は未実装です" );
    }

    #region Inner functions
    [DllImport("FocusTalkExt.dll", CharSet=CharSet.Ansi)]
    static extern Int32 FT_OutputVoiceText( string text, Int32 flags );

    [DllImport("FocusTalkExt.dll")]
    static extern bool FT_StopSound();
    #endregion
}

/// <summary>
/// 95Reader を単純なスピーカーとして使うためのクラス
/// </summary>
public class XpReaderSpeaker : ISpeaker
{
    static ISpeaker _Instance = null;

    XpReaderSpeaker()
    {}

    /// <summary>
    /// 唯一のインスタンスを取得
    /// </summary>
    public static ISpeaker Instance
    {
        get
```

```
    {
        if( _Instance == null )
        {
            _Instance = new XpReaderSpeaker();
        }
        return _Instance;
    }
}

/// <summary>
/// 95Reader をスピーカーとして使えるか判定
/// </summary>
/// <returns>使える場合は true</returns>
public static bool IsAlive()
{
    try
    {
        // API DLL の関数を正常に P/Invoke できるか確認
        SoundMessage( "", 0 );

        // 95Reader のウィンドウの存在を確認
        if( FindWindow("95Reader",0) == 0 )
        {
            return false;
        }
    }
    catch( DllNotFoundException )
    {
        return false;
    }

    return true;
}

/// <summary>
/// 文字列を読み上げさせる
/// </summary>
/// <param name="message">読み上げさせる文字列</param>
public void Speak( string message )
{
    const int PF_NATURAL_PROOF     = 0x40;
    int flag = 0;

    if( message.Length == 1
        || (0 < message.Length && Char.IsPunctuation(message, 0)) )
    {
        flag |= PF_NATURAL_PROOF;
    }

    SoundMessage( message, flag );
```

```
        }

        /// <summary>
        /// 読み上げを停止
        /// </summary>
        public void Stop()
        {
            SoundStop();
        }

        /// <summary>
        /// 読み上げに「間」を挿入する
        /// </summary>
        public void Wait()
        {
            SoundMessage( "、", 2 );
        }

        #region Inner Functions
        [DllImport("user32.dll", EntryPoint="FindWindowW", CharSet=CharSet.Unicode)]
        static extern Int32 FindWindow( string windowClassName, int windowText );

        [DllImport("SOUNDP.dll", CharSet=CharSet.Ansi)]
        static extern Int32 SoundMessage( string text, Int32 flags );

        [DllImport("SOUNDP.dll")]
        static extern bool SoundStop();
        #endregion
    }
}

/*
Version History

[v1.2.1] 2007-02-24
・XPReaderSpeaker を改善
・各スピーカを Singleton に

[v1.2.0] 2007-02-21
・ISpeaker.Wait を追加

[v1.1.0] 2007-02-20
・JawsSpeaker.Stop の実装を変更（前は環境により効果無かった）
・JAWS 7.1 の制御に対応

[v1.0.1] 2007-01-09
・他モジュールへの依存性を無くした

[v1.0.0] 2006-11-30
・リリース
```

```
*/

/*********************************************************
Copyright (c) 2006-2007 YAMAMOTO Suguru

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
*********************************************************/
```