



ATOK12/ATOK11/ATOK10 for Windows

AMET アプリケーション開発マニュアル

* 「ATOK12」「ATOK11」「ATOK10」は、株式会社ジャストシステムの著作物であり、
「ATOK12」「ATOK11」「ATOK10」にかかる著作権その他の権利はすべて
株式会社ジャストシステムに帰属します。

* 「ATOK」は株式会社ジャストシステムの登録商標です。

* その他、記載された製品名は各社の商標または登録商標です。

(C) 1998 株式会社ジャストシステム

ご使用条件

株式会社ジャストシステム（以下「弊社」）は、以下に定めるドキュメント、サンプルプログラム〔ソース/実行形〕、開発用ファイル（以下総称して「本技術情報」）を受領するお客様（以下「受領者」）が下記の条件にご同意されることを条件として、本技術情報の使用を許諾いたします。受領者が本技術情報のご使用を開始された場合は、下記条件に同意いただいているものとみなします。（本使用条件は、ドキュメントの本文、および開発用ファイル、サンプルプログラムの「License.txt」にも記載されています。）

ドキュメント : ATOK12/ATOK11/ATOK10 for Windows
AMET アプリケーション開発マニュアル
サンプルプログラム : AMET サンプルアプリケーション
開発用ファイル : (該当するものではありません)

記

1 定義

「アプリケーション」とは、本使用条件に従い本技術情報を使用して受領者が開発するプログラムまたはソフトウェア製品などをいいます。

2 受領者は、受領者個人または受領者の属する法人などの内部的な範囲において、個人的または商業的とを問わず、アプリケーション開発を目的とする場合に限り、以下に定める範囲で本技術情報を使用することができます。

- (1) 本技術情報を、閲覧、ダウンロード、複製して使用すること。
- (2) サンプルプログラムのソースコードについては、改変し、アプリケーションに組み込むこと。
- (3) アプリケーションを第三者に頒布すること。

3 受領者は以下の行為を行わないものとします。

- (1) 本使用条件に定める以外の目的での本技術情報の使用。
- (2) 本技術情報の一部または全部の、第三者への頒布、販売、再使用許諾、送信、Web サイトなどネットワークサーバへの掲示。
- (3) ドキュメント、開発用ファイルの一部または全部の、改変、リバースエンジニアリング。
- (4) 本技術情報を利用することによる、弊社 ATOK シリーズ競合製品の開発。
- (5) アプリケーションの開発、評価および使用の際に、真正品以外の ATOK または弊社製品を使用すること。

4 受領者はアプリケーションを頒布する際、有償であるか否かを問わず、以下を遵守するものとします。

- (1) 弊社の書面による許諾なく、弊社の社名や製品名等のロゴを使用しないこと。
- (2) アプリケーションが弊社製品に対応する旨を表示したり、マニュアル等で商品内容や操作を説明したりする場合を除き、弊社の社名および商品名を表示しないこと。
- (3) 前項に従って、製品名を記載する場合は、「ATOK」が弊社の登録商標である旨をあわせて表示すること。
- (4) アプリケーションについて弊社による推奨、監修、品質保証等がある、もしくは、アプリケーションが弊社から販売されているかの誤認を与えるような表示を行わないこと。
- (5) アプリケーションの頒布により生じた第三者との紛争に関しては、受領者の責任と負担により解決するものとし、弊社に損害を与えないこと。

5 本技術情報にかかる著作権その他の権利は、弊社に帰属するものであり、受領者は、本技術情報に付されている著作権、商標権その他の知的財産権の表示を削除、変更することはできません。

6 弊社は、

- (1) 本技術情報は現状有姿で提供され、その品質および機能が完全であることおよび受領者の使用目的に適合することを保証するものではなく、本技術情報についての瑕疵担保責任および保証責任を一切負いません。本技術情報の選択導入は受領者の責任で行っていただき、本技術情報の使用およびその結果についても同様とします。
- (2) 本技術情報およびアプリケーションの使用または使用不能から生ずる直接的または間接的損害については一切責任を負いません。

7 弊社は受領者に対し、本技術情報に関するサポートは一切行いません。

8 弊社は、受領者が本使用条件のいずれかの条項に違反した場合または弊社の著作権を侵害した場合には、本技術情報の受領者による使用を終了させることができます。その場合、受領者は、受領者が有する本技術情報及びその複製物を必ず破棄して下さい。

9 本技術情報は随時変更されることがあります。

以 上

目 次

1 . 概 要	1
2 . AMET の仕組み	2
2 - 1 . ATOK ウィンドウ	2
2 - 2 . 文字コードについて	2
3 . AMET サーバ	3
3 - 1 . プロパティとメソッド	3
3 - 2 . 必須プロパティ	4
3 - 3 . 必須メソッド	4
3 - 4 . 任意プロパティ	5
3 - 5 . 文節情報	6
3 - 6 . WM_AMET_NOTIFY メッセージ	6
4 . AMET サーバの登録	7
5 . AMET サーバの終了	8
6 . Visual Basic 4.0 による AMET サーバ作成例	9
6 - 1 . プロジェクトの作成	9
6 - 2 . フォームの設計	10
6 - 3 . Sub Main モジュールの作成	11
6 - 4 . クラスモジュールの作成	11
6 - 5 . プロパティとメソッドの実装	12
6 - 6 . ATOK への通知	17
6 - 7 . プロジェクトの保存と EXE ファイルの生成	18
6 - 8 . AMET サーバの登録	18
6 - 9 . ATOK との連携テスト	18
7 . Visual C++ 4.0 による AMET サーバ作成例	19
7 - 1 . プロジェクトワークスペースの作成	19
7 - 2 . ダイアログの編集	21
7 - 3 . プロパティ、メソッドの作成	22
7 - 4 . ドキュメントクラスの編集	24
7 - 5 . ビュークラスの編集	30
7 - 6 . メンバ関数の編集	31
7 - 7 . ビルドと実行テスト	33
7 - 8 . ATOK との連携テスト	33

1 . 概 要

本書は、ATOK12 for Windows(*1)用 AMET(*2)アプリケーションの開発方法について説明する。

AMET は ATOK のための拡張機能の仕組みである。ATOK は AMET の起動時に未確定文字列の読みと表記、文節情報を送信する。AMET はこれらの情報を使って処理をし、必要であれば ATOK に確定文字列を送信することができる。

AMET アプリケーションは、ATOK とは別のプロセスとして動作する OLE オートメーションサーバとして実装される。

(*1)本書で説明する AMET アプリケーションが動作する ATOK のバージョンは次のとおり

- ・ ATOK12 for Windows の各バージョン
- ・ ATOK11 for Windows の各バージョン
- ・ ATOK10 for Windows の各バージョン

(*2)AMET ... ATOK Multi Engine Transfer
「エイメット」と発音する。

ATOK11 までの AMET アプリケーション開発マニュアルをお使いの方へ

...この仕様書での変更点

- ・ 内容に変更はありません。
- ・ 対象バージョンに ATOK12 for Windows を追加。

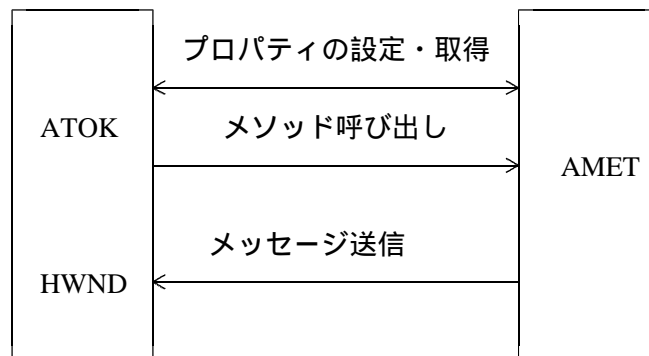
2 . AMET の仕組み

AMET のプログラムは OLE オートメーションサーバとして動作し、ATOK は OLE オートメーションのクライアントとして動作する。

ATOK はキー操作、またはメニューによる操作で AMET を起動する要求を受けると、INI ファイル(*3)の情報をもとに対応する AMET プログラムを起動する。そして、未確定文字列の読み、表記、文節情報を OLE オートメーションのプロパティとメソッドの機能を使って送信する。

(*3)ATOK12 の場合... ATOK12W.INI
 ATOK11 の場合... ATOK11W.INI
 ATOK10 の場合... ATOK10W.INI

AMET が ATOK に確定文字列を返すときは、AMET から ATOK に送信するデータがあることを通知する。OLE オートメーションにはサーバからクライアントに通知する標準の機能がないため、AMET では Windows のメッセージ送信の機能を使って通知している。



2 - 1 . ATOK ウィンドウ

上でも説明したように、AMET から ATOK に確定文字列を返すためには、ATOK にメッセージを送信している。ATOK は AMET を起動するときに AMET からのメッセージを受信するためのウィンドウを作成し、そのウィンドウハンドルを AMET に渡している。このウィンドウを『ATOK ウィンドウ』と呼ぶ。

ATOK は、ATOK ウィンドウに AMET からの通知メッセージが送信されると、AMET からプロパティを使って確定文字列を取得し、アプリケーションに送信する。

2 - 2 . 文字コードについて

OLE オートメーションでやり取りされる文字列(BSTR)は、Unicode の文字列として扱われる。Windows95 で使われる文字コードはシフト JIS だが、ATOK と AMET の間での文字コードは Unicode を使うものとする。

3 . AMET サーバ

AMET の機能を備えた OLE オートメーションサーバプログラムを『AMET サーバ』と呼ぶ。AMET サーバは要求されるプロパティとメソッドを実装していれば、どのような処理系で作成してもよい。本ドキュメントでは、Visual Basic 4.0 と Visual C++ 4.0 による AMET サーバの作成方法について説明してある。

3 - 1 . プロパティとメソッド

AMET サーバには次に示すプロパティとメソッドを実装することができる。プロパティとメソッドには必ず実装しなければならない必須プロパティ、必須メソッドと、必要ないときは実装しなくてもよい任意プロパティ、任意メソッドがある。

必須プロパティ		
AmetYomi	BSTR	読み文字列
AmetHyoki	BSTR	表記文字列
必須メソッド		
AmetStart(LONG)		データ送信通知
任意プロパティ		
AmetResult	BSTR	ATOKに返す確定文字列
AmetQuit	LONG	AMETサーバ終了フラグ
AmetBYomi	LONG[]	読み文節情報
AmetBYomiLen	LONG	読み文節情報のサイズ
AmetBHyoki	LONG[]	表記文節情報
AmetBHyokiLen	LONG	表記文節情報のサイズ
任意メソッド		
AmetSetBYomi(LONG, LONG)		読み文節情報の設定
AmetSetBHyoki(LONG, LONG)		表記文節情報の設定

3 - 2 . 必須プロパティ

必須プロパティは、ATOK が AMET サーバを呼び出すときに必ず設定するプロパティである。AMET サーバにこれらのプロパティのうちどれか一つでも実装されていないときは AMET の呼び出しは行われない。

3 - 2 - 1 . AmetYomi

AmetYomi は、未確定文字列の読みを設定する。

3 - 2 - 2 . AmetHyoki

AmetHyoki は、未確定文字列の表記を設定する。

3 - 3 . 必須メソッド

必須プロパティと同様、必須メソッドも ATOK が AMET サーバを呼び出すときに必ず実行するメソッドである。AMET サーバにこれらのメソッドのうちどれか一つでも実装されていないときは AMET の呼び出しは行われない。

3 - 3 - 1 . AmetStart

```
void AmetStart(LONG hWnd);
```

AMET サーバにプロパティの設定が完了したことを通知するメソッド。

引数の hWnd は、AMET サーバが ATOK にデータを送信するときの通知先のウィンドウ(ATOK ウィンドウ)のハンドルを与える。

3 - 4 . 任意プロパティ

任意プロパティは、AMET サーバが必要であれば実装するプロパティである。これらのプロパティが実装されていないときは ATOK はデフォルトの動作をする。

3 - 4 - 1 . AmetResult

AMET サーバが ATOK に確定文字列を返すときは、このプロパティを実装し、文字列を設定する。ATOK はここに設定された文字列を確定文字列としてアプリケーションに送信する。

このプロパティが実装されていないときは ATOK は確定文字列の設定をしない。

3 - 4 - 2 . AmetQuit

AMET サーバから ATOK に通知があった後、AMET サーバを終了するかどうかを表すプロパティ。TRUE(0)のときは AMET サーバは終了し、FALSE (= 0)のときは終了しない。

このプロパティが実装されていないときは TRUE が設定されているものとみなし、AMET サーバは終了する。

3 - 5 . 文節情報

AMET サーバが読みと表記の文節情報を必要とするとき、文節情報プロパティを実装することができる。これらのプロパティとメソッドは任意である。

3 - 5 - 1 . AmetBYomi, AmetBHyoki

文節情報を格納するためのデータ領域でプロパティではない。LONG の配列 (LONG[100]) として実装する。このデータ領域に値を設定する場合は、後述する AmetSetBYomi()、AmetSetBHyoki() メソッドを用いる。

3 - 5 - 2 . AmetBYomiLen, AmetBHyokiLen

それぞれ AmetBYomi, AmetBHyoki のうち、有効なデータの数を与えるプロパティ。

このプロパティが実装されていないときは文節情報の設定は行わない。

3 - 5 - 3 . AmetSetBYomi, AmetSetBHyoki

```
void AmetSetBYomi(LONG idx, LONG data);  
void AmetSetBHyoki(LONG idx, LONG data);
```

文節情報を設定するためのメソッド。AmetBYomi、AmetBHyoki の代わりとして使う。ATOK はこのメソッドを繰り返し呼び出し、配列の idx で与えられるインデックスの要素に data で指定された値を設定する。

このメソッドが実装されていないときは文節情報の設定は行わない。

3 - 6 . WM_AMET_NOTIFY メッセージ

AMET サーバが ATOK に確定文字列を返すときは、AmetStart の引数として渡された ATOK ウィンドウのハンドルに WM_AMET_NOTIFY メッセージを送信する。ATOK ウィンドウはこのメッセージを受信すると、AMET サーバの AmetResult、AmetQuit の各プロパティを調べる。

AmetResult が存在するときは、プロパティから確定文字列を取得し、アプリケーションに送信する。また AmetQuit が存在するときは、その値によって AMET サーバを終了する。

WM_AMET_NOTIFY は WM_USER + 100 として定義される。

4 . AMET サーバの登録

AMET サーバは OLE オートメーションサーバとして作成される。ユーザの環境で作成した AMET サーバを使えるようにするためには、次のステップが必要である。

- ・ AMET サーバをレジストリに登録する。
- ・ ATOK の.INI ファイル(*4)に AMET サーバの ProgID を登録する。

(*4)ATOK12 の場合... ATOK12W.INI
ATOK11 の場合... ATOK11W.INI
ATOK10 の場合... ATOK10W.INI

ATOK は.INI ファイルに登録された ProgID から対応する AMET サーバを起動するので、INI ファイルへの登録が必須である。通常.INI ファイルへの登録は AMET サーバのインストール時に行う。

```
[AMETX] ; X=1--9、0  
名称=候補リストに表示される名前  
ID=AMETサーバのProgID  
未入力起動=しない/する
```

5 . AMET サーバの終了

ATOK は AMET サーバが起動されている間 OLE オートメーションの接続をしたままになっている。だからユーザの手で AMET サーバを終了させてしまうと ATOK の動作も不安定になってしまう。3 - 6 . でも説明したように、ATOK は WM_AMET_NOTIFY メッセージが送信されたときに AmetQuit プロパティの存在を調べ、AMET サーバの終了を管理する。

そのため、AMET サーバは OLE オートメーションから起動されたときは手動で終了できなくする必要がある。その代わりに、AMET サーバが終了したいときは AmetQuit プロパティを TRUE にした WM_AMET_NOTIFY メッセージを ATOK に送信すればよい。

6 . Visual Basic 4.0 による AMET サーバ作成例

Visual Basic 4.0 (VB4)を使って AMET サーバを作る方法について説明する。なお、OLE オートメーションサーバの作り方に関しては、Books Online の Professional Edition Documentation - Creating OLE Servers が参考になる。

ここでは実際に VB4 で AMET サーバを作成し、その手順をまとめた。作成する AMET サーバ(AmetVB)は AMET の基本機能を備えたサーバである。

6 - 1 . プロジェクトの作成

VB4 で OLE サーバを作成する場合、通常プロジェクトとは違った設定が必要である。次の手順で OLE サーバ用のプロジェクトを作成する。

手順 1 . VB4 を起動し、新規プロジェクトを作成する。

手順 2 . 【ツール - オプション】を選択してオプションダイアログを開き、プロジェクトタブ を選択する。

手順 3 . スタートアップフォームを Sub Main にする。

手順 4 . プロジェクト名に AmetVB と入力する。

手順 5 . 実行開始モードを OLE サーバ にする。

手順 6 . 互換 OLE サーバは指定しない。

手順 7 . アプリケーションの説明に説明を入力する。これは AMET では使われない。

手順 8 . [OK] をクリックする。

6 - 2 . フォームの設計

フォームにコントロールを貼り付け、図 1 . のようにする。

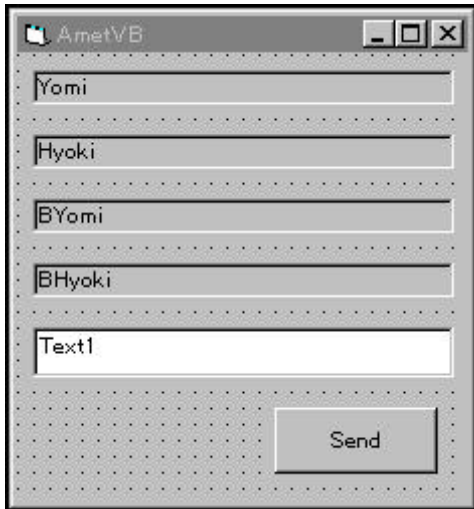


図 1 .

コントロールのプロパティを次の表のように設定する。

コントロールの種類	プロパティ	値
Form	Name Caption	AmetVBForm AmetVB
Label	Name Caption BorderStyle	AmetYomi Yomi 1 - 実線
Label	Name Caption BorderStyle	AmetHyoki Hyoki 1 - 実線
Label	Name Caption BorderStyle	AmetBYomi BYomi 1 - 実線
Label	Name Caption BorderStyle	AmetBHyoki BHyoki 1 - 実線
TextBox	Name	AmetResult
CommandButton	Name Caption	SendResult Send

6 - 3 . Sub Main モジュールの作成

プロジェクトでスタートアップフォームを Sub Main にしたので、Sub Main のモジュールを作成しなければならない。

手順 1 .【挿入 - 標準モジュール】を選択する。

手順 2 . プロパティウィンドウが表示されていないときは【表示 - プロパティウィンドウ】を指定し、プロパティウィンドウを表示する。

手順 3 . プロパティウィンドウの Name プロパティ を modDSMain に設定する。

手順 4 . modDSMain モジュールに次のコードを追加する。

```
Sub Main()  
    'フォームを表示する  
    AmetVBForm.Show  
End Sub
```

6 - 3 - 1 . フォームの表示と消去について

VB4 で作成した OLE オートメーションサーバは、OLE オートメーションによって起動されたときにはフォームを表示しない。そのため、プログラムの起動時にフォームを表示するためのコードを書かなければならない。AmetVB では SubMain の中でフォームを表示させたが、それ以外の場所でもよい。

また、プログラムの終了時にフォームが表示されていると正常に終了できないので、フォームを消去後終了しなければならない。一般的にはクラスの終了処理 Class_Terminate() の中で、フォームを消去して、終了するのがよい。

6 - 4 . クラスモジュールの作成

一般に VB4 で作成する OLE サーバは、最低一つは公開されたクラスモジュールを持つ。AMET サーバは原則として一つだけのクラスモジュールを持つこととする。クラスモジュールは次の手順で作成する。

手順 1 .【挿入 - クラスモジュール】を選択する。

手順 2 . プロパティウィンドウで Name プロパティ に AmetClass を設定する。

手順 3 . Public プロパティ を True に設定する。

手順 4 . Instancing プロパティ を 1 - CreatableSingleUse にする。

6 - 5 . プロパティとメソッドの実装

AmetVB はすべてのプロパティとメソッドを実装する。

VB4 でプロパティを実装するときは、クラスモジュールに Public 変数を定義するか、Property 手続きを定義する。メソッドを実装するときは、Public Sub か Public Function を定義する。

6 - 5 - 1 . Public 変数プロパティの実装

Public 変数によるプロパティを実装するときは、クラスモジュールの declarations セクションで変数の宣言をする。

AmetClass の declarations セクションに次の変数宣言を記述する。

```
Option Explicit
Public AmetYomi As String
Public AmetHyoki As String
Public AmetQuit As Long
Public AmetBYomiLen As Long
Public AmetBHyokiLen As Long
```

6 - 5 - 2 . Property 手続きによるプロパティの実装

AmetResult は他のプロパティとは異なり、リードオンリープロパティなので、手続きによるプロパティとして実装する。それには次の手順が必要である。

手順 1 . AmetClass の declarations セクションに次の宣言を追加する。

```
Private AmetResultString As String
```

手順 2 . 【挿入 - プロシージャ】を選択する。

手順 3 . 名前ボックスに AmetResult と入力する。

手順 4 . 種類に Property プロシージャ を指定する。

手順 5 . スコープに Public プロシージャ を指定する。

手順 6 . [OK] をクリックする。

手順 7 . Public Property Get AmetResult() と Public Property Let AmetResult() という 2 つのプロパティプロシージャが生成される。これらを次のように変更する。AmetResult はリードオンリーなので、Public Property Let AmetResult() の方は何もしない。

```
Public Property Get AmetResult() As String
    AmetResultString = AmetVbForm.AmetResult.Text
    AmetResult = AmetResultString
End Property
```

```
Public Property Let AmetResult(vNewValue As String)
    '何もしない。
End Property
```

6 - 5 - 3 . AmetStart メソッドの実装

手順 1 . フォームの declarations セクションに次の変数を定義する。

```
' AMET通知ウィンドウハンドル  
Public AtokWindow As Long
```

手順 2 . AmetClass のコード (declarations セクション) を表示する

手順 3 . 【挿入 - プロシージャ】を選択する。

手順 4 . 名前ボックスに AmetStart と入力する。

手順 5 . 種類に Sub プロシージャ を指定する。

手順 6 . スコープに Public プロシージャ を指定する。

手順 7 . [OK] をクリックする。

手順 8 . プロシージャが生成されるので、次のように修正する。

```
Public Sub AmetStart(hwnd As Long)  
    AmetVBForm.AtokWindow = hwnd  
    ' フォームにデータを表示する。  
    AmetVBForm.AmetYomi.Caption = AmetYomi  
    AmetVBForm.AmetHyoki.Caption = AmetHyoki  
    AmetVBForm.AmetBYomi.Caption = Str(AmetBYomiLen) + ":" +  
        Str(AmetBYomi(0)) + "," + Str(AmetBYomi(1)) + "," +  
        Str(AmetBYomi(2))  
    AmetVBForm.AmetBHyoki.Caption = Str(AmetBHyokiLen) + ":" +  
        Str(AmetBHyoki(0)) + "," + Str(AmetBHyoki(1)) + "," +  
        Str(AmetBHyoki(2))  
End Sub
```

6 - 5 - 4 . AmetSetBYomi、AmetSetBHyoki メソッドの実装

AmetClass の declarations セクションに次の変数宣言を追加する。

```
Private AmetBYomi(100) As Long  
Private AmetBHyoki(100) As Long
```

AmetStart と同様の手順で AmetSetBYomi、AmetSetBHyoki の 2 つのメソッドを実装し、次のように修正する。

```
Public Sub AmetSetBYomi(idx As Long, data As Long)  
    If idx <= 100 Then AmetBYomi(idx) = data  
End Sub
```

```
Public Sub AmetSetBHyoki(idx As Long, data As Long)  
    If idx <= 100 Then AmetBHyoki(idx) = data  
End Sub
```

6 - 5 - 5 . クラスの初期化と終了処理

VB4 のクラスモジュールには、C++のコンストラクタとデストラクタに対応して、Class_Initialize()と Class_Terminate()という関数がある。AmetClass のそれぞれの関数を次のようにする。

```
Private Sub Class_Initialize()  
    AmetYomi = ""  
    AmetHyoki = ""  
    AmetQuit = True  
    AmetBYomiLen = 0  
    AmetBHyokiLen = 0  
    AmetResultString = 0  
    Dim i As Integer  
    For i = 0 To 100  
        AmetBYomi(i) = 0  
        AmetBHyoki(i) = 0  
    Next  
    AmetVBForm.AtokWindow = 0  
End Sub
```

```
Private Sub Class_Terminate()  
    AmetVBForm.Hide  
    End  
End Sub
```

Class_Terminate()では、メインフォームを消去後、End を実行してプログラムを終了させなければならない。

6 - 6 . ATOK への通知

ATOK に通知するときは、AmetStart() で渡されたウィンドウハンドルに WM_AMET_NOTIFY (= WM_USER + 100) メッセージを送信する。

6 - 6 - 1 . API ビューワー

VB4 で Windows の API を使用するためには、API 関数の宣言が必要である。これは VB4 に附属する API ビューワーを使うことで生成できる。

手順 1 . API ビューワーを起動する。

手順 2 . 種類コンボボックスで 関数 を選択する。

手順 3 . 【ファイル - テキストファイルの読み込み】を選択する。

手順 4 . Win32api.txt を読み込む。データベースに変換するか、と聞かれるので、よく使う場合は変換しておく。

手順 5 . 有効な項目 リストボックスから、PostMessage を検索し、[追加] をクリックする。

手順 6 . 次に 種類コンボボックスを 定数 に変更する。

手順 7 . 有効な項目 リストボックスから、WM_USER を検索し、[追加] をクリックする。

手順 8 . [コピー] をクリックする。

手順 9 . API ビューワーを終了する。

API ビューワーで [コピー] をクリックしたときに、API の宣言がクリップボードに読み込まれているので、Sub Main モジュールの declarations セクションに貼り付ける。それに続けて WM_AMET_NOTIFY の定義を追加する。その結果 Sub Main モジュールの declarations セクションは次のようになる。

```
Declare Function PostMessage Lib "user32" Alias "PostMessageA"  
    (ByVal hwnd As Long, ByVal wParam As Long, ByVal lParam As Long,  
     ByVal IParam As Long) As Long  
Public Const WM_USER = &H400  
Public Const WM_AMET_NOTIFY As Long = WM_USER + 100
```

6 - 6 - 2 . イベントプロシージャの追加

フォームの [Send] ボタンをダブルクリックして、イベントプロシージャ SendResultClick を生成する。プロシージャを次のように修正して、ATOK にメッセージを送信する処理を追加する。

```
Private Sub SendResult_Click()
    If AtokWindow <> 0 Then PostMessage AtokWindow, WM_AMET_NOTIFY, 0, 0
End Sub
```

6 - 7 . プロジェクトの保存と EXE ファイルの生成

ファイル - 名前を付けてプロジェクトの保存 を選択し、プロジェクトを適当なディレクトリに保存する。プロジェクト名を変更してもプロジェクトファイルの名前はデフォルトの Project1 になっているので、AmetVB に変更する。

【ファイル - EXE ファイルの作成】を選択する。EXE ファイルの名前を聞いてくるので、AmetVB.EXE とする。

6 - 8 . AMET サーバの登録

VB4 で作成した AMET サーバは、コマンドラインで /REGSERVER オプションを付けて実行することでレジストリに登録される。登録を解除するときは、/UNREGSERVER オプションを付けて実行する。

6 - 8 - 1 . ProgID について

VB4 で作成した OLE サーバの ProgID は、プロジェクト名とクラス名の組み合わせになる。AmetVB の場合の ProgID は AmetVB.AmetClass である。

6 - 9 . ATOK との連携テスト

ATOK から呼び出せるようにするために ATOK の INI ファイル(*5)に次のように登録する。

```
[AMET1]
名称=AmetVBサーバ
ID=AmetVB.AmetClass
未入力起動=しない
```

メモ帳などで ATOK を起動し、未確定文字列のある状態で Shift + F10 キーを押してメニューを表示し、AMET メニューを起動する。そこから 1 番の AMET を選択すると AmetVB が起動し、読みと表記が表示される。エディットボックスに任意の文字列を入力して Send ボタンをクリックすると、その文字列が確定文字列としてアプリケーションに送信され、AmetVB は終了する。

(*5)ATOK12 の場合... ATOK12W.INI
ATOK11 の場合... ATOK11W.INI
ATOK10 の場合... ATOK10W.INI

7 . Visual C++ 4.0 による AMET サーバ作成例

Visual C++ 4.0 (VC4)と AppWizard を使って AMET サーバを作る方法について説明する。

OLE オートメーションサーバの作成方法については Books Online - Visual C++ Books - Tutorials - OLE Automation Server Tutorial の AutoClik が参考になる。

ここでは実際に VC4 で AMET サーバを作成し、その手順をまとめた。作成する AMET サーバ(AmetVC)は AMET の基本機能を備えたサーバである。

AmetVC は ATOK から起動されると読み、表記、文節情報を表示する。エディットボックスに文字列を入力して Send ボタンをクリックすると、その文字列を確定文字列として ATOK に送信し、即座に終了する。

AmetVC は VC4 の AppWizard で作成する。その際、画面の設計が簡単になるので、ビュークラスには CFormView を使うことにした。

7 - 1 . プロジェクトワークスペースの作成

- 手順 1 . 【ファイル - 新規作成】で表示される 新規作成ダイアログボックス で プロジェクトワークスペース を選択し、[OK]ボタンをクリックする。
- 手順 2 . 新規プロジェクトワークスペースダイアログボックスが表示されるので、タイプ で MFC AppWizard (EXE) を選択し、プロジェクトワークスペース名 に AmetVC と入力して [作成]ボタンをクリックする。
- 手順 3 . ステップ 1 の 作成するアプリケーションのタイプ で SDI を指定する。
- 手順 4 . ステップ 3/6 の OLE サポートの選択 で、OLE オートメーション を指定する。
- 手順 5 . ステップ 4/6 では、ドッキングツールバー、初期ステータスバー、印刷および印刷プレビュー のチェックをはずす。特に最初の 2 つがチェックされていると、起動時の表示がおかしくなってしまう。
- 手順 6 . ステップ 4/6 で[高度な設定]ボタンをクリックして、高度なオプションダイアログボックスを開き、ファイルタイプ ID ボックスに入力された Prog ID を確認する。変更してもよいが、ここではデフォルトの AmetVC.Document をそのまま使う。
- 手順 7 . ステップ 6/6 の AppWizard で作成される新規アプリケーションクラス リストボックスで CAmetVCView を選択する。基本クラス が CView になっているので、CFormView に変更する。
- 手順 8 . [終了]をクリックすると、新規プロジェクト情報ダイアログボックスが表示されるので、[OK]をクリックしてソースを生成する。

7 - 1 - 1 . プロジェクトオプションの変更

AmetVC のプログラム中で RTTI を使用する。RTTI はデフォルトでは無効になっているので、次の手順で有効にする。

手順 1 . 【ビルド - 設定】でプロジェクト設定ダイアログボックスを開く。

手順 2 . C/C++ タブ を選択し、カテゴリコンボボックスで C++ 言語 を選択する。

手順 3 . ランタイムタイプ情報(RTTI)を行うチェックボックスをクリックし、RTTI を有効にする。

手順 4 . [OK]をクリックし、ダイアログボックスを閉じる。

7 - 2 . ダイアログの編集

プログラムのメイン画面は IDD_AMETVC_FORM で与えられるダイアログなので、これを編集する。

プロジェクトワークスペースウィンドウの Resource View タブ を選択し、AmetVC リソース - Dialog - IDD_AMETVC_FORM をダブルクリックしてダイアログエディタを起動する。

ダイアログを図 2 . のように変更する。



図 2 . AmetVC のダイアログ

それぞれのコントロールのプロパティは次のように設定する。

種類	ID	キャプション	その他
Static	IDC_AMETYOMI	Yomi	Border
Static	IDC_AMETHYOKI	Hyoki	Border
Static	IDC_AMETBYOMI	BYomi	Border
Static	IDC_AMETBHYOKI	BHyoki	Border
Edit	IDC_AMETRESULT		
Button	IDC_SENDRESULT	Send	

編集が終われば、ダイアログエディタを終了する。

7 - 3 . プロパティ、メソッドの作成

AmetVC は、AMET サーバが持つことのできるすべてのプロパティとメソッドを実装する。

【表示 - ClassWizard】を実行し、OLE オートメーションタブ を選択する。クラス名で CAmetVCDoc クラスを選択すると、[メソッドの追加]、[プロパティの追加]ボタンが有効になる。

7 - 3 - 1 . プロパティの追加

AmetYomi プロパティを追加するときは次のようにする。プロパティの インプリメント には メンバ変数 と 取得/設定メソッド の 2 種類を指定できるが、AmetVC ではすべてのプロパティを メンバ変数 として実装する。

手順 1 . [プロパティの追加]ボタンをクリックしてプロパティの追加ダイアログボックスを開く。

手順 2 . 外部名 に AmetYomi と入力する。

手順 3 . タイプ に CString を指定する。

手順 4 . 他はデフォルトのままにする。

手順 5 . [OK]ボタンをクリックする。

同様に残りのプロパティも追加する。プロパティの属性は次のようになる。

外部名	タイプ	変数名	通知関数
AmetYomi	CString	m_ametYomi	OnAmetYomiChanged
AmetHyoki	CString	m_ametHyoki	OnAmetHyokiChanged
AmetResult	CString	m_ametResult	OnAmetResultChanged
AmetQuit	long	m_ametQuit	OnAmetQuitChanged
AmetBYomiLen	long	m_ametBYomiLen	OnAmetBYomiLenChanged
AmetBHyokiLen	long	m_ametBHyokiLen	OnAmetBHyokiLenChanged

7 - 3 - 2 . メソッドの追加

AmetStart プロパティを追加するときは次のようにする。

手順 1 . [メソッドの追加] ボタンをクリックしてメソッドの追加ダイアログボックスを開く。

手順 2 . 外部名 に AmetStart と入力する。

手順 3 . 内部名に AmetStart と入力されるので、そのままにする。

手順 4 . 戻り値の型に void を指定する。

手順 5 . パラメータリスト で引数を指定する。

a . 変数名 の下でクリックすると、引数名を入力できるようになるので、hWnd と入力する。

b . タイプ の下でクリックして、引数の型を long にする。

手順 6 . [OK] ボタンをクリックする。

同様に残りのメソッドも追加する。メソッドの属性は次のようになる。

外部名	内部名	戻り値の型	パラメータリスト	
			変数名	タイプ
AmetStart	AmetStart	void	hWnd	long
AmetSetBYomi	AmetSetBYomi	void	idx	long
			data	long
AmetSetBHyoki	AmetSetBHyoki	void	idx	long
			data	long

7 - 4 . ドキュメントクラスの編集

メソッドの追加が完了したところで ClassWizard の [コード編集] をクリックし、ソースの編集画面を表示する。

7 - 4 - 1 . メンバ変数の追加

最初に Class Wizard で自動的に追加されないメンバ変数を追加する。

m_hAtokWindow を追加するときの手順を次に示す。その前にプロジェクトウインドウの Class View タブをクリックして表示を切り換える。

手順 1 . プロジェクトウインドウの AmetVC クラス - CAmetVCDoc を選択する。

手順 2 . マウスの右ボタンを押して表示されるメニューから【変数の追加】を選択するとメンバ変数の追加ダイアログが表示される。

手順 3 . 変数の型 に HWND と入力する。

手順 4 . 変数宣言 に m_hAtokWindow と入力する。

手順 5 . アクセス は Private を指定する。

手順 6 . [OK] をクリックする。

同様に次に示すメンバ変数を追加する。

変数の型	変数宣言	アクセス	説明
HWND	m_hAtokWindow	Private	ATOKウインドウ
long	m_AmetBYomi [100]	Private	読み文節情報
long	m_AmetBHyoki [100]	Private	表記文節情報

7 - 4 - 2 . メンバ関数の追加

Class Wizard で自動的に追加されないメンバ関数を追加する。

通常のメンバ関数 `AmetNotify()` を追加するときの手順を次に示す。

- 手順 1 . プロジェクトウィンドウの `AmetVC` クラス - `CAmetVCDoc` を選択する。
- 手順 2 . マウスの右ボタンを押して表示されるメニューから【関数の追加】を選択するとメンバ関数の追加ダイアログが表示される。
- 手順 3 . 関数型 に `void` と入力する。
- 手順 4 . 関数宣言に `AmetNotify(void)` と入力する。
- 手順 5 . アクセス は `Public` を指定する。
- 手順 6 . [OK] をクリックする。

`inline` メンバ関数を追加するときは、直接ヘッダファイルを編集する。その手順を次に示す。

- 手順 7 . プロジェクトウィンドウの `AmetVC` クラス - `CAmetVCDoc` を選択する。
- 手順 8 . マウスの右ボタンを押して表示されるメニューから【定義の表示】を選択すると、`CAmetVCDoc` のヘッダファイルが表示される。
- 手順 9 . `CAmetVCDoc` のクラス定義の最初の部分に『//アトリビュート』という行があるので、その行以降を次のように変更する。
先頭に『>』のついた行を追加する。

```
// アトリビュート
public:
    void AmetNotify(void);
> CString GetYomi(void) const      {return(m_ametYomi);}
> CString GetHyoki(void) const    {return(m_ametHyoki);}
> void    SetResult(const CString str)  {m_ametResult = str;}
> long    GetYomiLen(void) const  {return(m_ametBYomiLen);}
> long    GetHyokiLen(void) const {return(m_ametBHyokiLen);}
> long    GetBYomi(int idx) const {return(m_AmetBYomi[idx]);}
> long    GetBHyoki(int idx) const  {return(m_AmetBHyoki[idx]);}
```

追加が終われば、ヘッダファイルを保存する。

ここで追加したメンバ関数について以下にまとめる。

関数型	関数宣言	アクセス	説明
void	AmetNotify(void)	Pub	ATOKにメッセージを送る
CString	GetYomi(void)	Pub	読み情報の取得
CString	GetHyoki(void)	Pub	表記情報の取得
void	SetResult(const CString str)	Pub	確定情報の設定
long	GetYomiLen(void)	Pub	読み文節情報長取得
long	GetHyokiLen(void)	Pub	表記文節情報長取得
long	GetBYomi(int idx)	Pub	読み文節情報取得
long	GetBHyoki(int idx)	Pub	表記文節情報取得

7 - 4 - 3 . メンバ関数の編集

追加したメンバ関数を編集する。以下の説明では先頭に 『 > 』をつけた行を追加する。以下で説明していないメンバ関数は ClassWizard が生成したままで変更しない。

関数の定義場所への移動方法について

関数を編集するとき、関数の定義されている場所に移動するには、ソースファイル上で探す以外に、次のような方法もある。

プロジェクトウィンドウで AmetVC クラス - CAmetVCDoc - AmetNotify() をダブルクリックすると、ソースファイルの AmetNotify() が定義されているところに移動することができる。

7 - 4 - 4 . コンストラクタ

CAmetVCDoc()は、追加したプロパティに関連したメンバ変数とメンバ変数の初期化をするように変更する。

```
CAmetVCDoc::CAmetVCDoc()
{
>  m_ametYomi = "";
>  m_ametHyoki = "";
>  m_ametResult = "";
>  m_hAtokWindow = 0;
>  m_ametQuit = FALSE;
>  m_ametBYomiLen = 0L;
>  m_ametBHyokiLen = 0L;
>  int i;
>  for(i = 0; i < 100; i++) m_AmetBYomi[i] = 0L;
>  for(i = 0; i < 100; i++) m_AmetBHyoki[i] = 0L;

  EnableAutomation();

  AfxOleLockApp();
}
```

7 - 4 - 5 . AmetStart()

AmetStart()では次の処理を追加する。

- ・引数として渡された ATOK ウィンドウのハンドルを保存する。
- ・ウィンドウを表示する。
- ・フレームウィンドウのサイズを調整する。

```
void CAmetVCDoc::AmetStart(long hWnd)
{
> m_hAtokWindow = (HWND)hWnd;
> // ウィンドウの表示
> POSITION pos = GetFirstViewPosition();
> CScrollView* pView = dynamic_cast<CScrollView*>(GetNextView(pos));
> if(pView != NULL)
> {
>     CFrameWnd* pFrameWnd = pView->GetParentFrame();
>     pFrameWnd->ActivateFrame(SW_SHOW);
>     // フレームウィンドウの大きさを調整する
>     // OLEオートメーションで起動されたときだけ有効
>     pView->ResizeParentToFit();
> }
}
```

ウィンドウの表示について

VC4 + AppWizard で作成した OLE オートメーションサーバは、オートメーションサーバとして起動されたときはウィンドウを表示しないため、自分で表示しなければならない。AMET サーバでは AmetStart()がウィンドウの表示をするタイミングとして適切なので、ここでウィンドウの表示をする。

フレームウィンドウのサイズ調整

AmetVC は CFormView クラスの派生クラスをビューとして使っている。このクラスはダイアログをフレームウィンドウのクライアントウィンドウとして使うため、フレームウィンドウのサイズをダイアログのサイズに合わせなければならない。

単独のプログラムの場合は、ビュークラスの OnInitialUpdate()でサイズ調整をするが、OLE オートメーションではその設定が反映されないため、ウィンドウの表示処理と同時にフレームウィンドウのサイズ調整もすることにした。

7 - 4 - 6 . AmetSetBYomi(), AmetSetBHyoki()

これらの関数は、次のように実装する。

```
void CAmetVCDoc::AmetSetBYomi(long idx, long data)
{
>   if(idx < 100)   m_AmetBYomi[idx] = data;
}

void CAmetVCDoc::AmetSetBHyoki(long idx, long data)
{
>   if(idx < 100)   m_AmetBHyoki[idx] = data;
}
```

7 - 4 - 7 . AmetNotify()

ATOK に WM_AMET_NOTIFY メッセージを送信するための関数。AmetQuit を TRUE にしているため、このメッセージを送ったあと ATOK は AmetVC を終了させる。

```
>const   int           WM_AMET_NOTIFY = WM_USER + 100;

void CAmetDVDoc::AmetNotify(void)
{
>   if(m_hAtokWindow != 0)
>   {
>       m_ametQuit = TRUE;
>       ::PostMessage(m_hAtokWindow, WM_AMET_NOTIFY, 0, 0);
>   }
}
```

7 - 5 . ビュークラスの編集

ビュークラスはイベントに対応する処理などを追加する。

7 - 5 - 1 . オーバーライドメンバ関数の追加

上位のクラスで定義されているメンバ関数をオーバーライドする。一例として、OnDraw()を定義する手順を次に示す。

手順 1 . ClassWizard を起動し、メッセージマップタブを選択する。

手順 2 . クラス名 で CAmetVCView を選択する。

手順 3 . オブジェクト ID リストボックスで CAmetVCView を選択する。

手順 4 . メッセージリストボックスで OnDraw を選択する。

手順 5 . [関数の追加] ボタンをクリックすると、OnDraw()の雛形が生成される。

同様に次に示すオーバーライドメンバ関数を追加する。

名前	説明
OnDraw	ビューの再表示。
OnInitialUpdate	ビューを最初に表示するときの処理。

7 - 5 - 2 . イベントハンドラの追加

ボタンが押されたときのイベントハンドラ関数を追加するには、次の手順で行う。

手順 1 . ClassWizard を起動し、メッセージマップタブを選択する。

手順 2 . クラス名 で CAmetVCView を選択する。

手順 3 . オブジェクト ID リストボックスで IDC_SENDRESULT を選択する。

手順 4 . メッセージリストボックスで BN_CLICKED を選択する。

手順 5 . [関数の追加] ボタンをクリックすると、イベントハンドラ関数の名前を尋ねてくるので、デフォルトの名前(OnSendResult)を変更せずに OK をクリックする。

7 - 6 . メンバ関数の編集

追加したメンバ関数を編集する。以下の説明では先頭に 『 > 』をつけた行を追加する。

7 - 6 - 1 . OnDraw

OnDraw()は、ビューが表示されるときに呼び出される。AMET の呼び出し時に設定された情報をドキュメントから取得してコントロールに表示する。

```
void CAmetVCView::OnDraw(CDC* pDC)
{
> CAmetVCDoc* pDoc = GetDocument();
> SetDlgItemText(IDC_AMETYOMI, pDoc->GetYomi());
> SetDlgItemText(IDC_AMETHYOKI, pDoc->GetHyoki());
> char buf[50];
> ::wsprintf(buf, "%2d: %2d, %2d, %2d", pDoc->GetYomiLen(),
> pDoc->GetBYomi(0), pDoc->GetBYomi(1), pDoc->GetBYomi(2));
> SetDlgItemText(IDC_AMETBYOMI, buf);
> ::wsprintf(buf, "%2d: %2d, %2d, %2d", pDoc->GetHyokiLen(),
> pDoc->GetBHyoki(0), pDoc->GetBHyoki(1), pDoc->GetBHyoki(2));
> SetDlgItemText(IDC_AMETBHYOKI, buf);
}
```

7 - 6 - 2 . OnInitialUpdate

CFormView クラスのウィンドウを表示するとき、自動的にウィンドウのサイズをダイアログのサイズに合せることができる。そのために OnInitialUpdate() ハンドラを追加する。

この関数は、AmetVC を単独で起動したときに有効であり、ATOK から OLE オートメーションで呼ばれたときは、ここの設定は反映されない。

```
void CAmetVCView::OnInitialUpdate()
{
    CFormView::OnInitialUpdate();

> ResizeParentToFit();
}
```

7 - 6 - 3 . OnSendResult

Send ボタンが押されたとき、エディットボックスに入力された文字列を ATOK に送る処理を追加する。

```
void CAmetVCView::OnSendresult()  
{  
> CString str;  
> GetDlgItemText(IDC_AMETRESULT, str);  
> CAmetVCDoc* pDoc = GetDocument();  
> pDoc->SetResult(str);  
> pDoc->AmetNotify();  
}
```

7 - 7 . ビルドと実行テスト

動作確認とレジストリに登録するため、ビルドして実行する。図 2 . のような画面が表示されればとりあえずは正常に動作している。

実行形を別のフォルダへ移す場合は、再度レジストリに登録する必要がある。

7 - 7 - 1 . ProgID について

VC4 で作成した OLE サーバの ProgID は、AppWizard のステップ 4/6 で [高度な設定] ボタンを押して開いた ファイルタイプ ID ボックスに入力された文字列が使われる。AmetVC の場合の ProgID は AmetVC.Document である。

7 - 8 . ATOK との連携テスト

ATOK から呼び出せるようにするために ATOK の INI ファイル(*6)に次のように登録し、再起動させる。

```
[AMET2]
名称=AmetVCサーバ
ID=AmetVC.Document
未入力起動=しない
```

メモ帳などで ATOK を起動し、未確定文字列のある状態で Shift + F10 キーを押してメニューを表示し、AMET メニューを起動する。そこから 2 番の AMET を選択すると AmetVC が起動し、読みと表記が表示される。エディットボックスに任意の文字列を入力して Send ボタンをクリックすると、その文字列が確定文字列としてアプリケーションに送信され、AmetVC は終了する。

(*6)ATOK12 の場合... ATOK12W.INI
ATOK11 の場合... ATOK11W.INI
ATOK10 の場合... ATOK10W.INI

以 上