

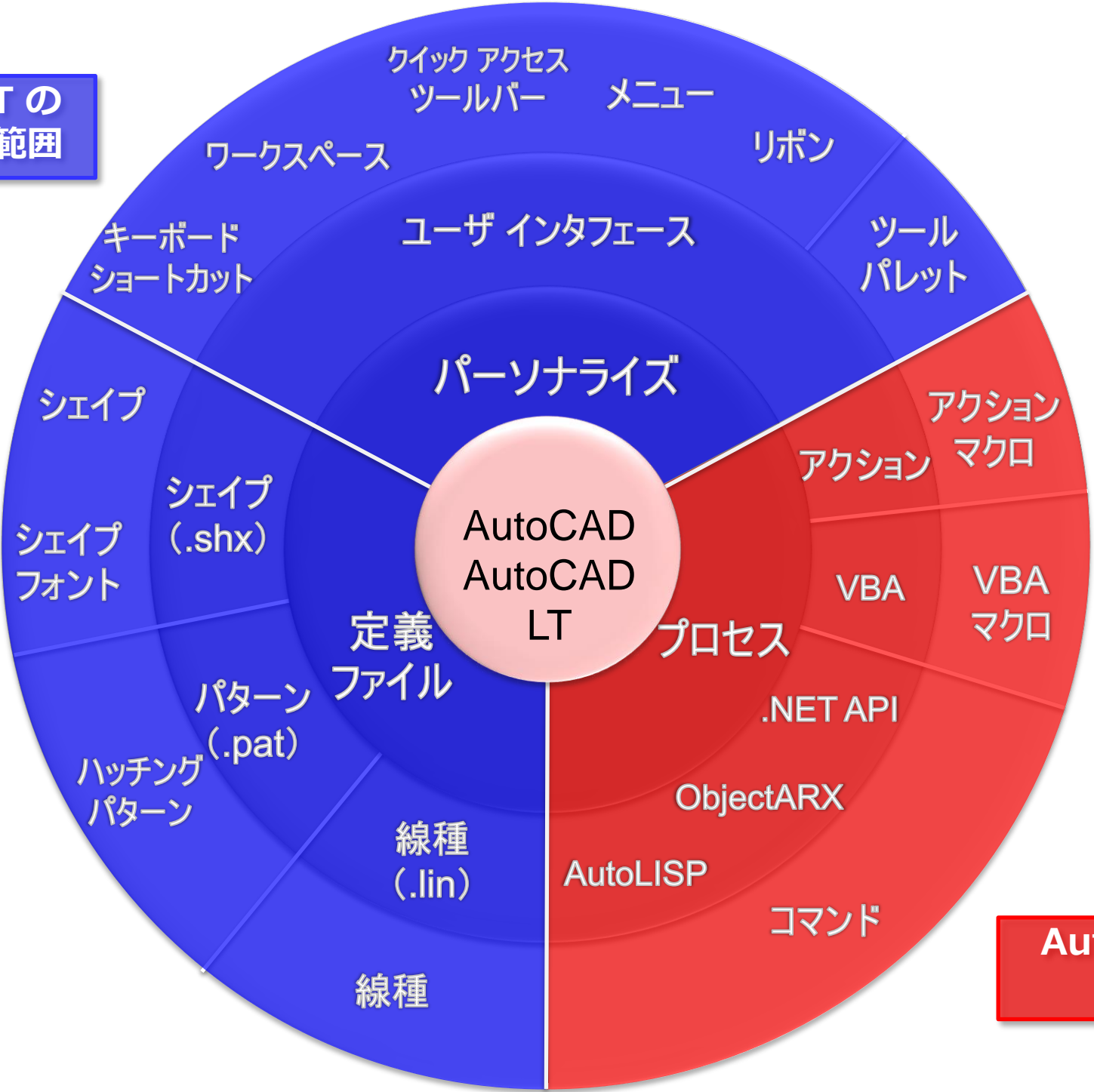
# AutoCAD 2022 .NET API トレーニング

オートデスク 株式会社  
Autodesk Developer Network

# AutoCAD のカスタマイズとは？

# カスタマイズ可能な項目

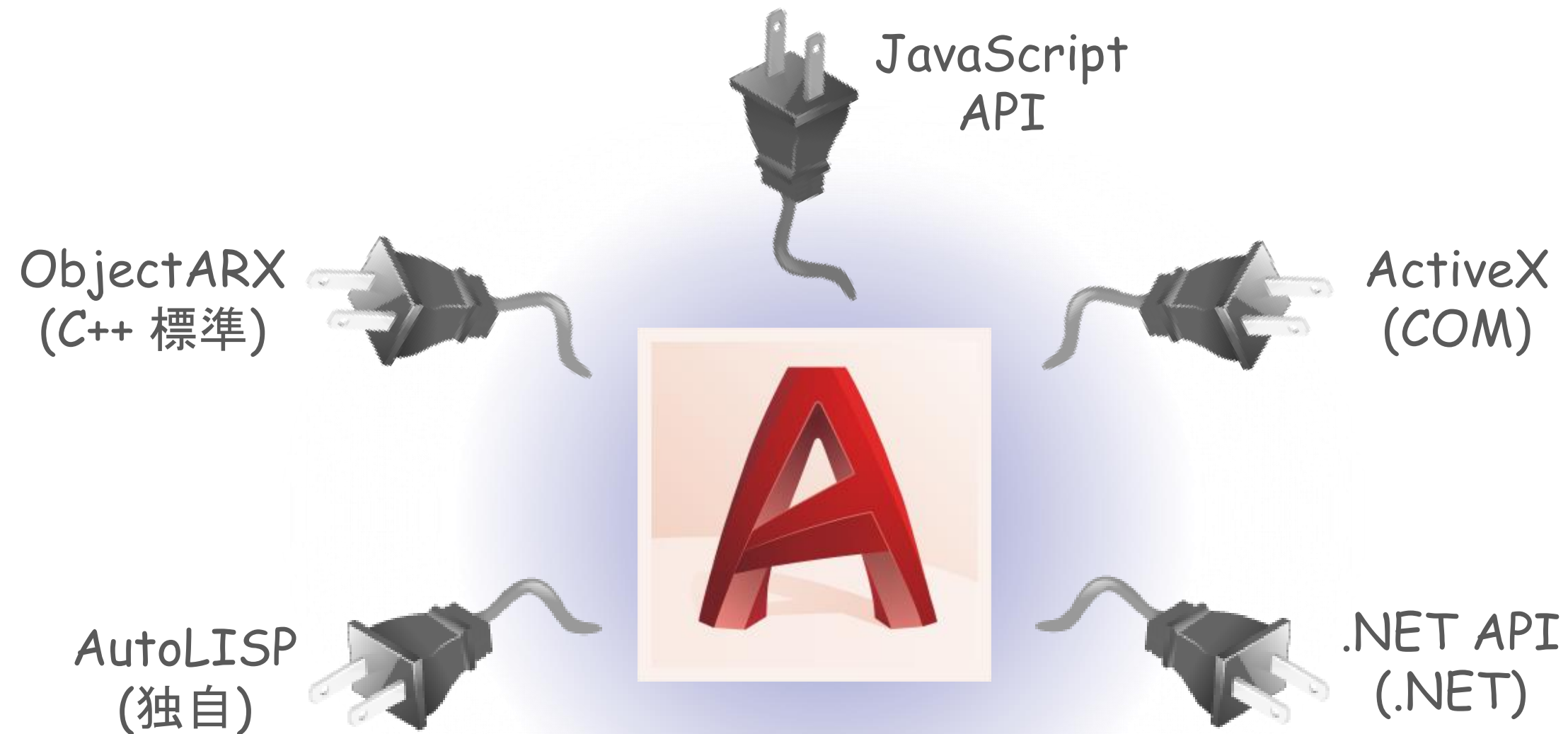
AutoCAD と AutoCAD LT の  
両方で可能なカスタマイズ範囲



AutoCAD でのみ可能な  
カスタマイズ範囲

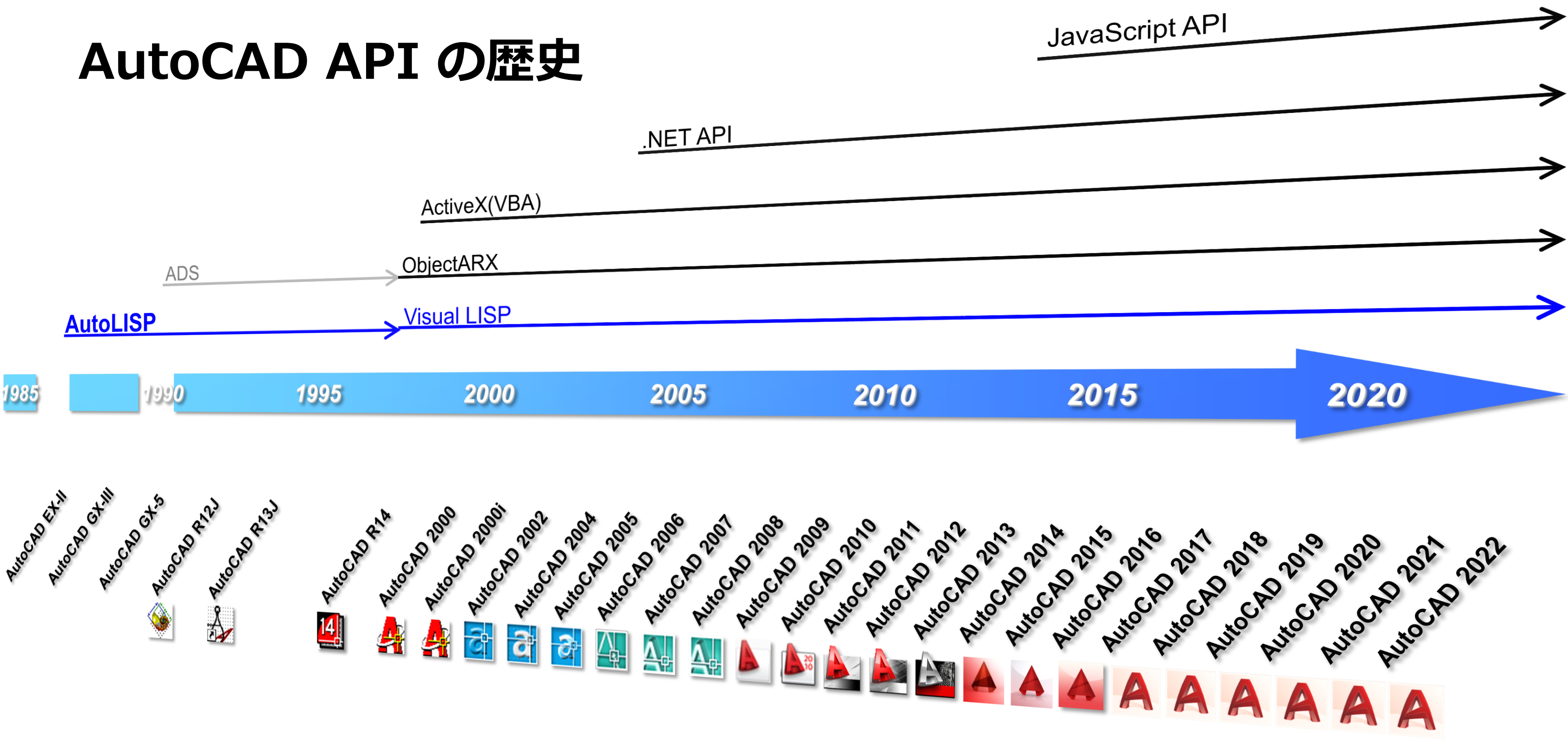
# AutoCAD API

- 柔軟で強力な Application Programming Interface

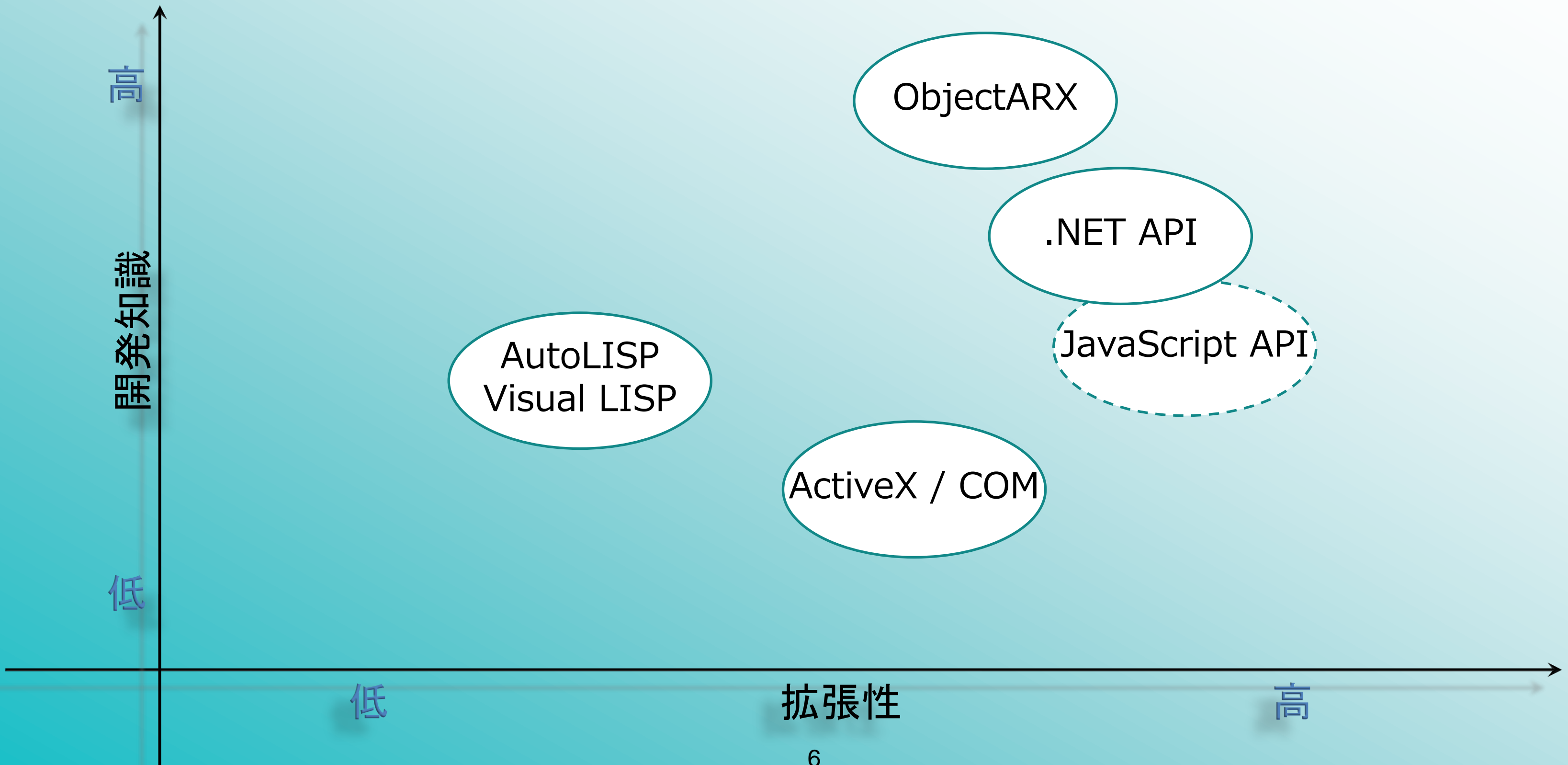




# AutoCAD API の歴史

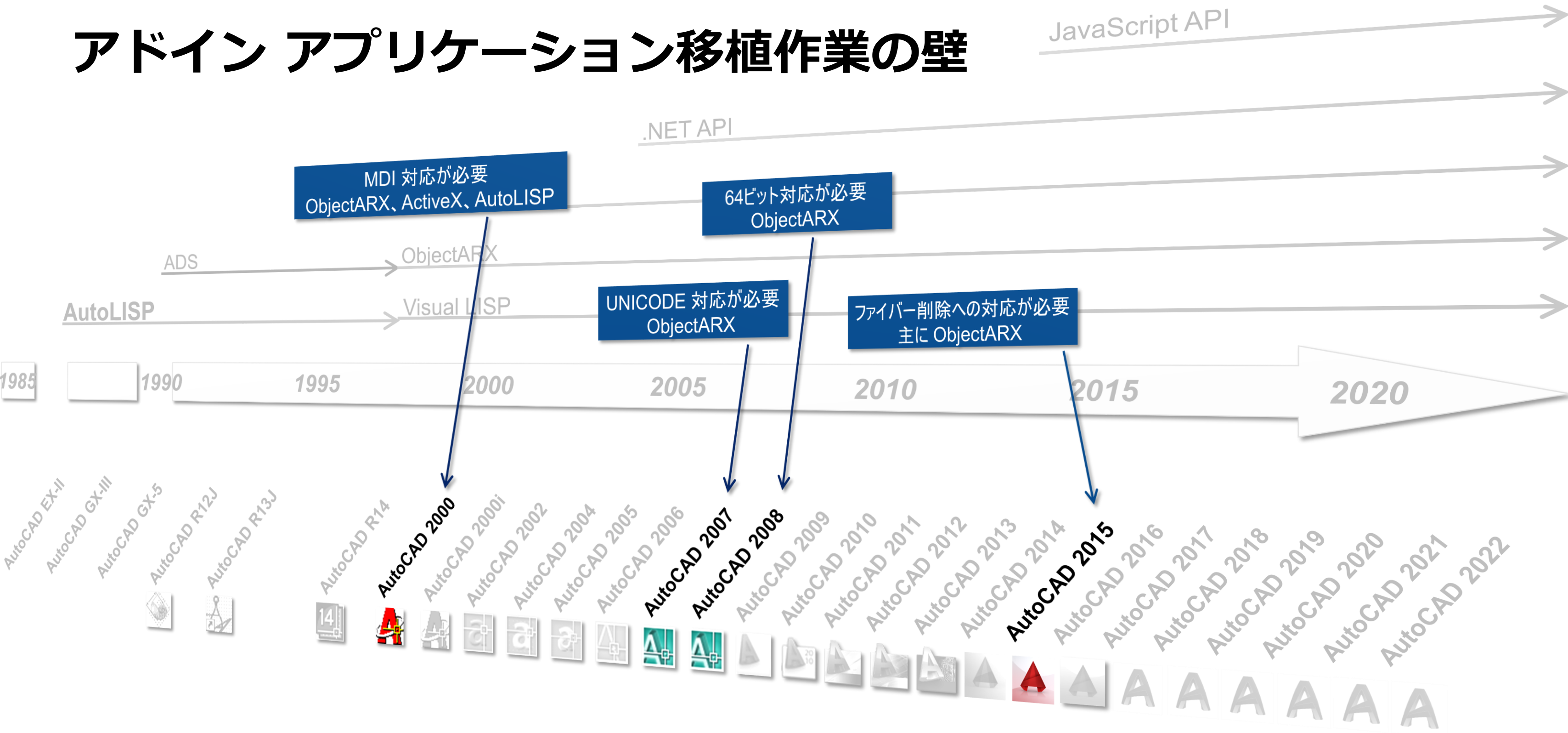


# AutoCAD API の位置付け



AutoCAD API 別の能力		Visual LISP	ActiveX/VBA	ObjectARX	.NET API
一般	グラフィカル オブジェクトの作成	○	○	○	○
一般	グラフィカル オブジェクトの編集	○	○	○	○
一般	非グラフィカル オブジェクトの作成	○	○	○	○
一般	非グラフィカル オブジェクトの編集	○	○	○	○
カスタム	オブジェクトの定義	—	—	○	—
カスタム	オブジェクトの作成	○	○	○	○
カスタム	オブジェクトの編集	○	○	○	○
	ファイルへの図面の保存	○	○	○	○
	既存図面を AutoCAD 上に開く	○	○	○	○
	既存図面をメモリ上に開く	—	—	○	○
	AutoCAD 上での新規図面の作成	○	○	○	○
	メモリ上での新規図面の作成	—	—	○	○
	オブジェクトの選択	○	○	○	○
	データ リスト操作	○	—	○	—
	コマンドの定義	○	○	○	○
	コマンドの発行	○	○	○	○
	リアクタ / イベント処理	○	○	○	○
	メニュー UI の実装	○	○	○	○
	ダイアログ UI の実装	○	○	○	○
	データベース アプリケーションとの連携	○	○	○	○
	他の SDK との連携	○	○	○	○
	幾何演算ライブラリ	—	—	○	○
	境界表示解析(B-Rep)	—	—	○	○
	COM インタフェースの実行	△	○	○	○
	オブジェクト指向プログラミング	7 ○	○	○	○

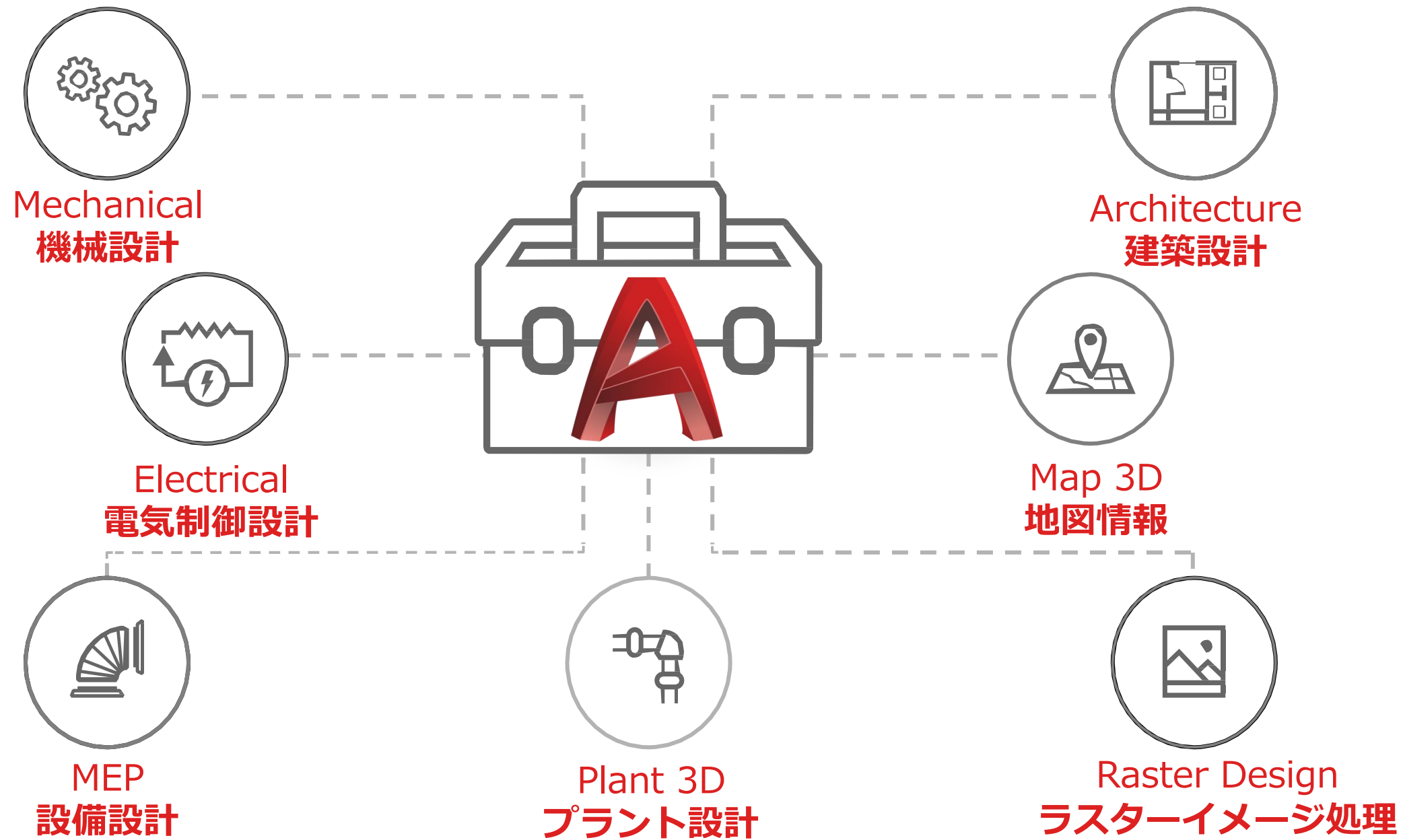
# アドイン アプリケーション移植作業の壁





# AutoCAD API により強力な機能拡張を実現

- 業界別オートデスク製品も AutoCAD API で拡張



# AutoCAD 業種別ツールセット

✓ 含まれる機能
AutoCAD
AutoCAD Architecture
AutoCAD Mechanical
AutoCAD Electrical
AutoCAD Map 3D
AutoCAD MEP (日本未発売)
AutoCAD Raster Design
AutoCAD Plant 3D
AutoCAD Mac (日本未発売)

✗ 含まれない機能
AutoCAD Civil 3D
AutoCAD LT
AutoCAD Advance Steel (日本未発売)
AutoCAD OEM (開発者専用)
AutoCAD P&ID (Plant 3D に統合)

- 業種別製品 API/SDK の提供方法に変更はありません

# AutoCAD ベース業種別製品固有の API

- 固有オブジェクト、固有操作等が可能な API の有無

製品	AutoLISP	ObjectARX	ActiveX/COM	.NET API
AutoCAD	○	○	○	○
AutoCAD Mechanical	×	△	△	×
AutoCAD Electrical	○	×	×	×
AutoCAD Map 3D	×	△	△	○
AutoCAD Civil 3D	×	×	○	○
AutoCAD Architecture	×	○	△	○
AutoCAD P&ID	×	○	×	○
AutoCAD Plant 3D	×	○	×	○
Raster Design	×	×	×	×

△：一部の固有オブジェクト、固有操作のみの API 制御のみ可能

※ AutoCAD が提供する 標準の API 4 種はどの業界別製品でも動作します

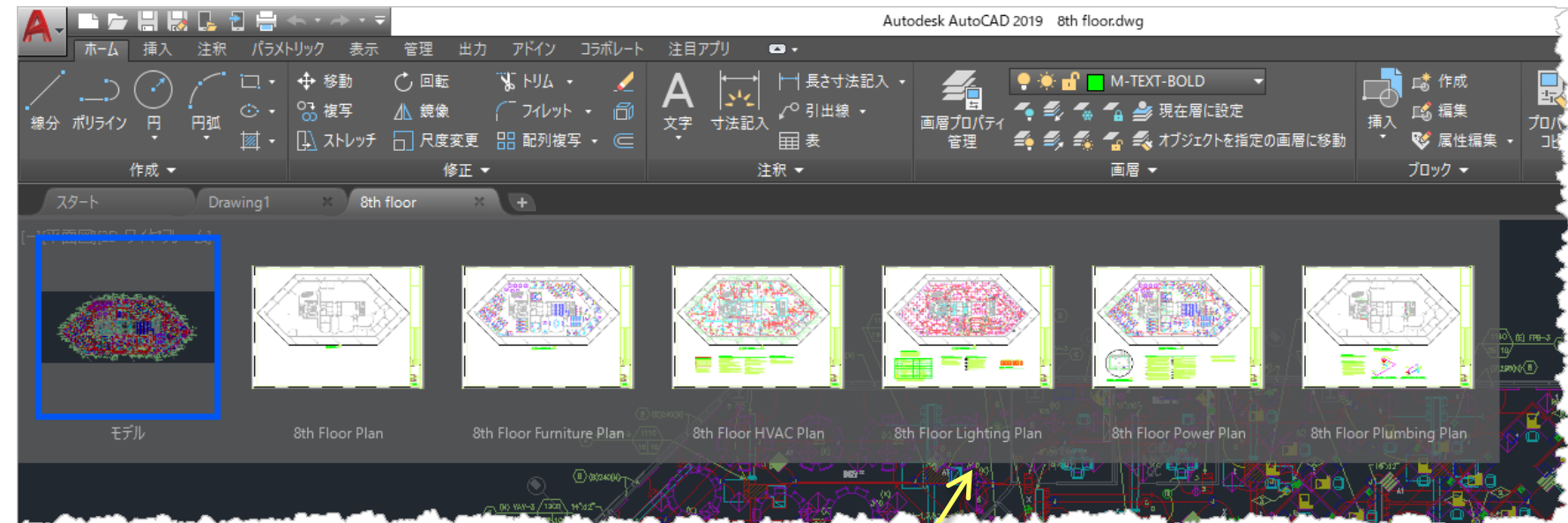
※ JavaScript API は全AutoCAD 製品で利用可能ですが業種固有機能の API はありません

# **.NET API の利点**

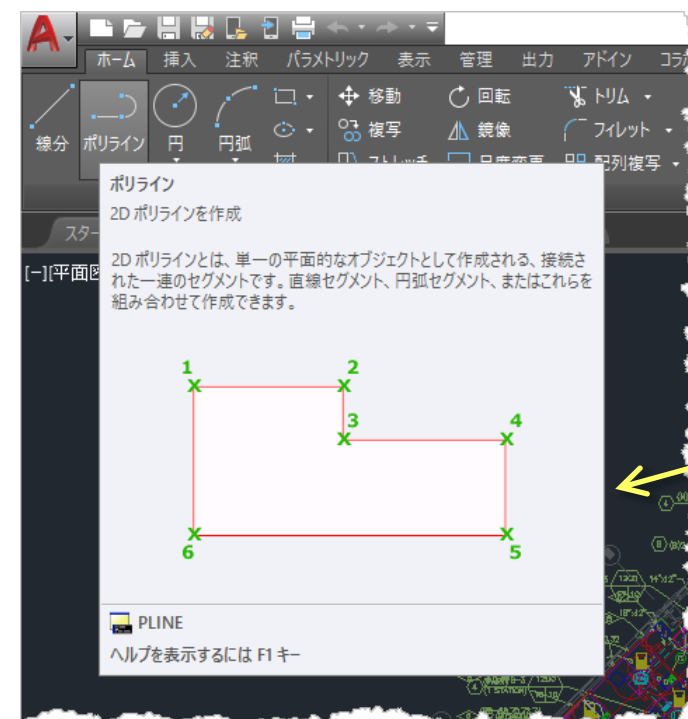


# オートデスク製品（AutoCAD）と .NET Framework

- AutoCAD 2005 : .NET Framework 1.1
- AutoCAD 2006 : .NET Framework 1.1
- AutoCAD 2007 : .NET Framework 2.0
- AutoCAD 2008 : .NET Framework 2.0
- AutoCAD 2009 : .NET Framework 3.0
- AutoCAD 2010 : .NET Framework 3.5
- AutoCAD 2011 : .NET Framework 3.5 SP1
- AutoCAD 2012 : .NET Framework 4.0
- AutoCAD 2013 : .NET Framework 4.0 Update1
- AutoCAD 2014 : .NET Framework 4.0 Update1
- AutoCAD 2015 : .NET Framework 4.5
- AutoCAD 2016 : .NET Framework 4.5
- AutoCAD 2017 : .NET Framework 4.6
- AutoCAD 2018 : .NET Framework 4.6
- AutoCAD 2019 : .NET Framework 4.7
- AutoCAD 2020 : .NET Framework 4.7
- AutoCAD 2021 : .NET Framework 4.8
- AutoCAD 2022 : .NET Framework 4.8



クイック ビュー レイアウト

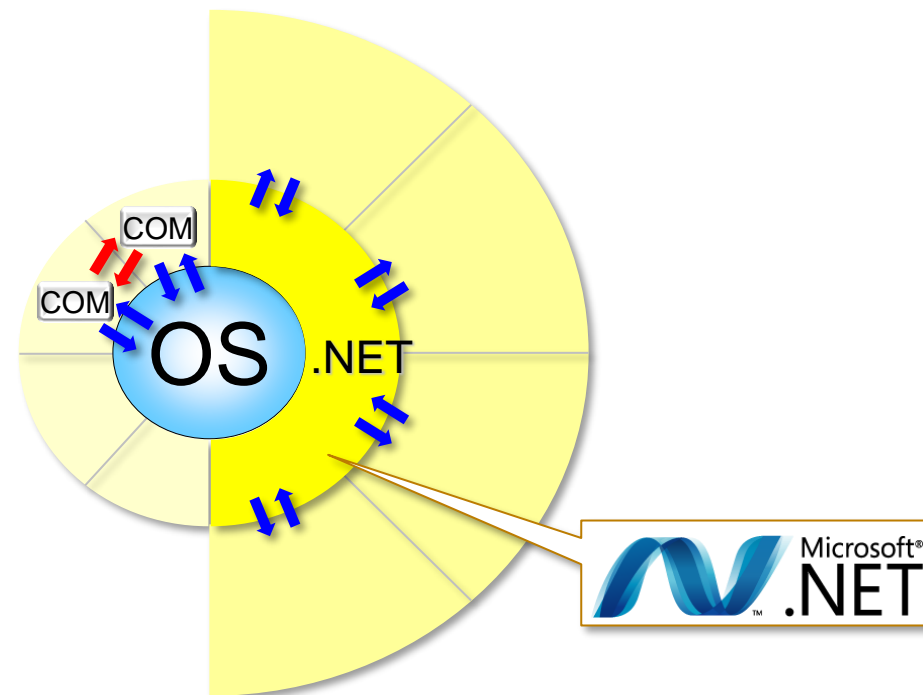


セカンダリ ツールチップ

# .NET API が利用するテクノロジー

## ■ .NET Framework

- Windows OS の一部として動作するインターネット時代の新しい基盤技術
- その中的一部分として ...
  1. サーバー：情報をオブジェクトとして公開する機能
  2. クライアント：公開されたオブジェクトを参照して再利用する機能
- 情報の公開や参照は OS やアプリケーション単位で **アセンブリ** が橋渡し

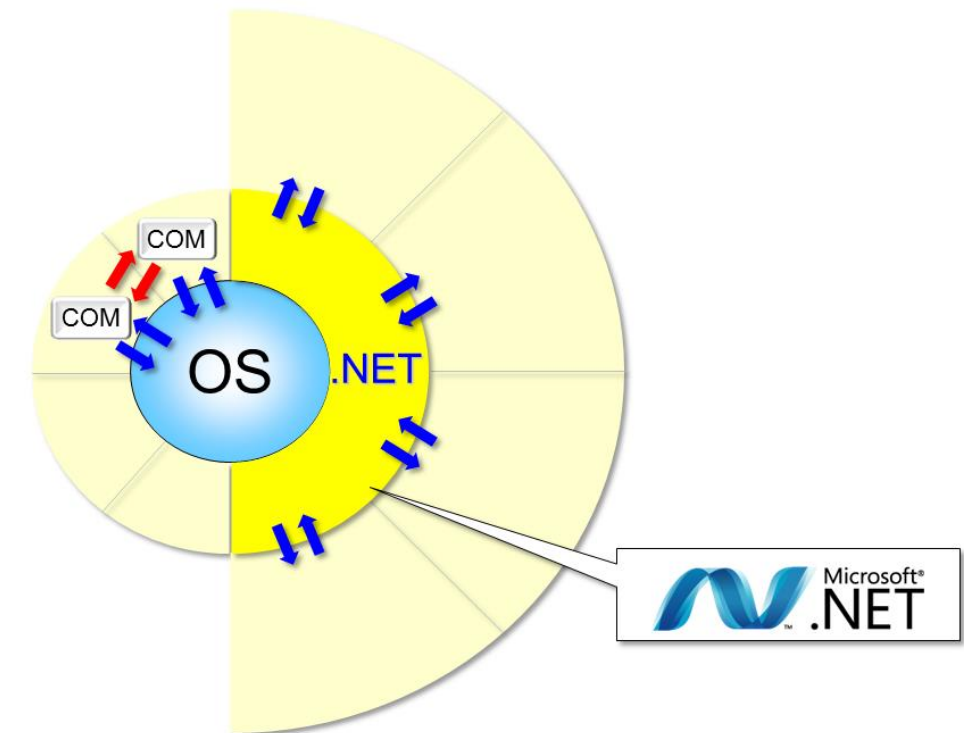


# .NET Framework の 2 つの側面

- .NET アプリケーションの実行環境としての側面

- 共通言語ランタイム

- Common Language Runtime (CLR)
    - .NET アプリケーションの実行に必須
    - 開発言語の選択が自由
    - 実行されるのは中間言語です



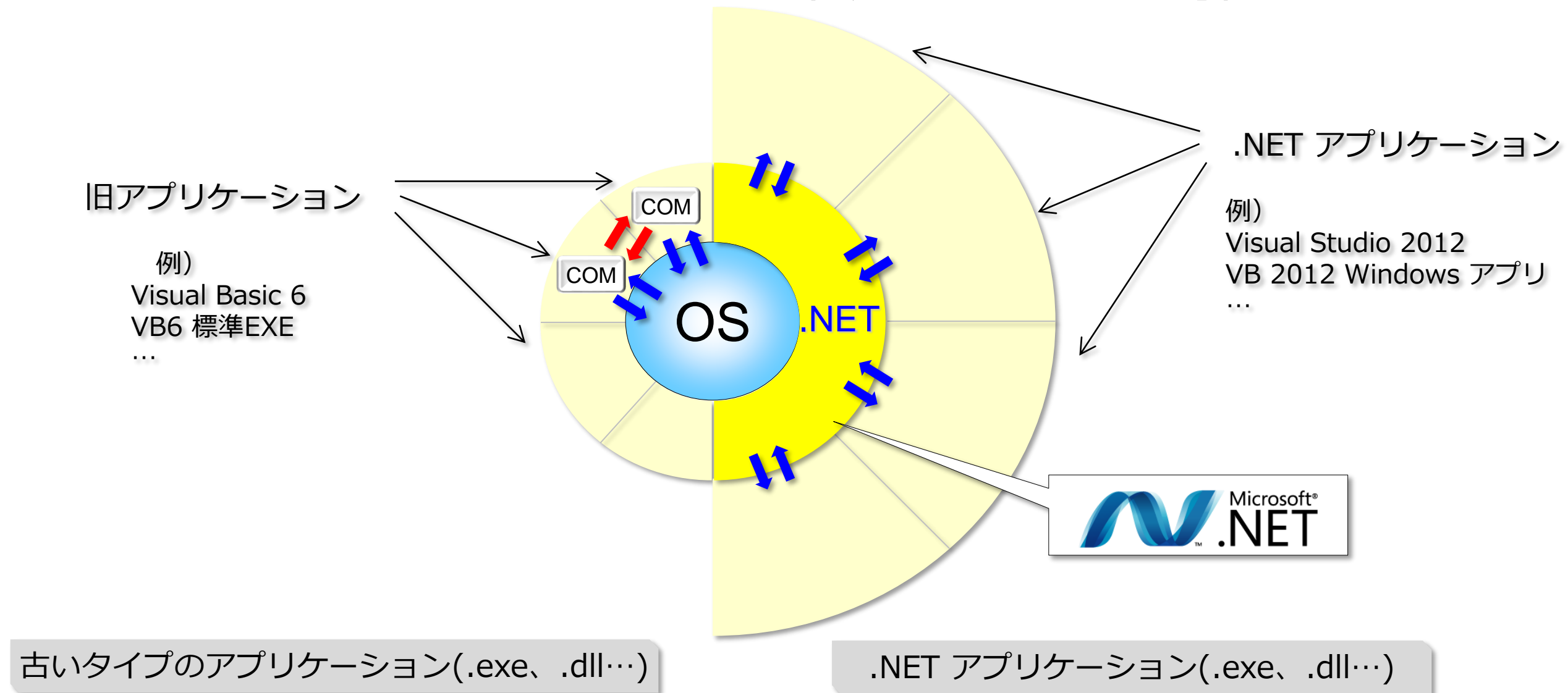
- .NET アプリケーションへの機能提供者としての側面

- .NET Framework クラス ライブラリ

- .NET アプリケーションを作成するための機能のかたまり
      - 開発者が使用するための機能
      - VBA で使う メソッド、プロパティ、イベントのようなもの

# 実行環境としての .NET Framework

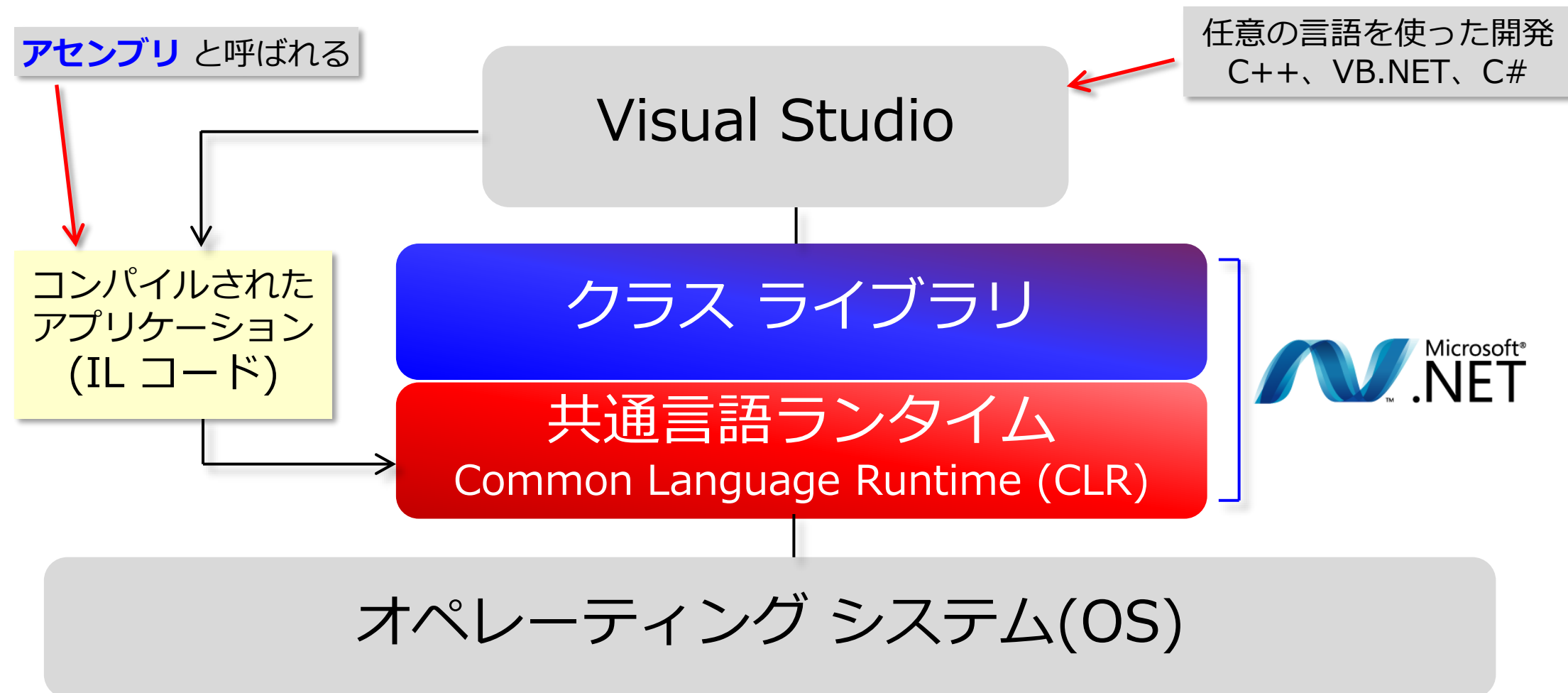
- OS に .NET アプリケーションの実行環境を与える…
  - .NET アプリケーションで COM を利用することも可能





# .NET アプリケーションの実行メカニズム 1

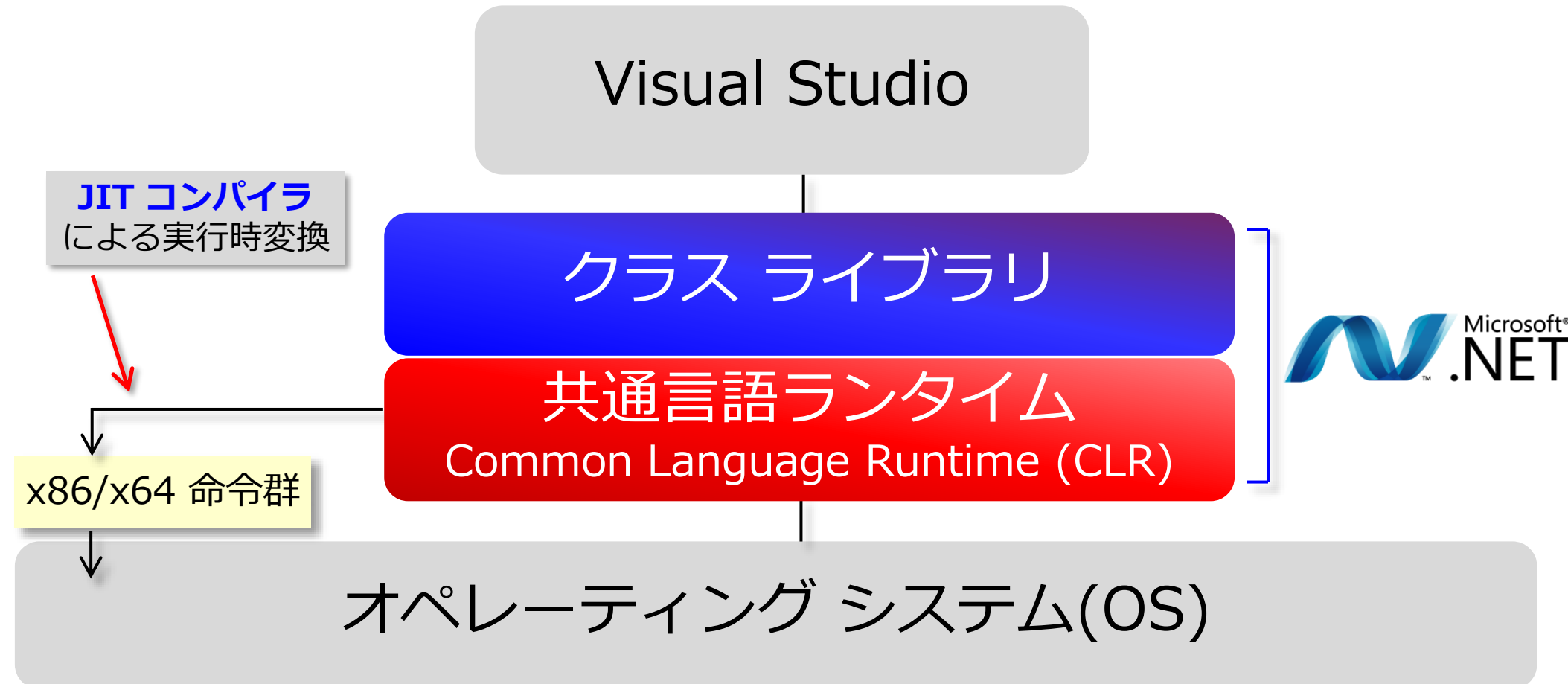
- ビルドされた .NET アプリケーションは中間コードに変換されます…



## .NET アプリケーションの実行メカニズム 2

- アセンブリは実行時に x86/64 バイナリにコンパイル、実行される

.NET Framework はプラットフォームの違いを吸収



# .NET から見たアプリケーションの種類

## ■ マネージ コード

- CLR によって管理、実行されるアプリケーション
- ネイティブな .NET アプリケーション
  - .NET Framework クラスライブラリで構築されている

## ■ アンマネージ コード

- CLR によって管理、実行されないアプリケーション
- 前世代のアプリケーション
  - Win32 API、MFC、COM などで構築されている

# .NET Framework クラス ライブラリ概要

- .NET Framework に含まれるオブジェクト関数群
  - 名前空間 という単位で公開されている
    - COM でいう オブジェクト
    - 同様にメソッドやプロパティを持つ
    - 公開するのは各アプリケーションでなく .NET Framework
    - 公開は タイプライブラリ 経由でなく アセンブリ ファイル 経由
- オブジェクト指向なオブジェクトで構成される
  - オブジェクト間は つながり を持ち階層構造で表現される
- 名前空間の表現方法
  - COM と同じようにつながり を ピリオド . で表現
  - <名前空間名>.<メソッド>、<名前空間名>.<プロパティ>



# .NET Framework クラス ライブラリの名前空間

## System.Web

Services  
Description  
Discovery  
Protocols  
Caching  
Configuration  
UI  
HtmlControls  
WebControls  
Security  
SessionState

Web サービスを含む Web 関連

## System.Windows.Forms

Design  
ComponentModel

Windows フォーム関連

## System.Drawing

Drawing  
Imaging  
Text

グラフィックス関連

## System.Data

ADO  
Design  
SQL  
SQLTypes

データベース (ADO.NET) 関連

## System.Xml

XSLT  
XPath  
XPathNavigator

XML 処理関連

## System

Collections  
Configuration  
Diagnostics  
Globalization  
IO  
Reflection  
Resources  
Security  
Text  
Threading  
Runtime  
Remoting  
Serialization

データ型、属性、数値演算など基本機能関連

# AutoCAD 2022 マネージ .NET API

- AutoCAD 固有クラスを公開するアセンブリ ファイル
  - [accoremgd.dll](#)
    - プラットフォーム共通機能
  - [acdbmgd.dll](#)
    - データベース サービス と DWG ファイル操作のみ
  - [acmgd.dll](#)
    - AutoCAD エディタ固有機能
  - [accui.dll](#)
    - メニューカスタマイズ固有機能
- AutoCAD インストール フォルダに格納されています
  - 例) C:\Program Files\AutoCAD 2022\
- 開発時は ObjectARX SDK にも同梱のアセンブリ ファイルを参照
  - 例) C:\ObjectARX 2021\inc\

# AutoCAD マネージ クラス ライブラリの名前空間

**Autodesk.AutoCAD.GraphicsInterface**

ベクトル描画機構関連

**Autodesk.AutoCAD.PlottingServices**

印刷サービス関連

**Autodesk.AutoCAD.Windows**

ユーザインタフェース関連

**Autodesk.AutoCAD.EditorInput**

選択、入力等ユーザ対話関連

**Autodesk.AutoCAD.Runtime**

例外処理など基本機能関連

**Autodesk.AutoCAD.Geometry**

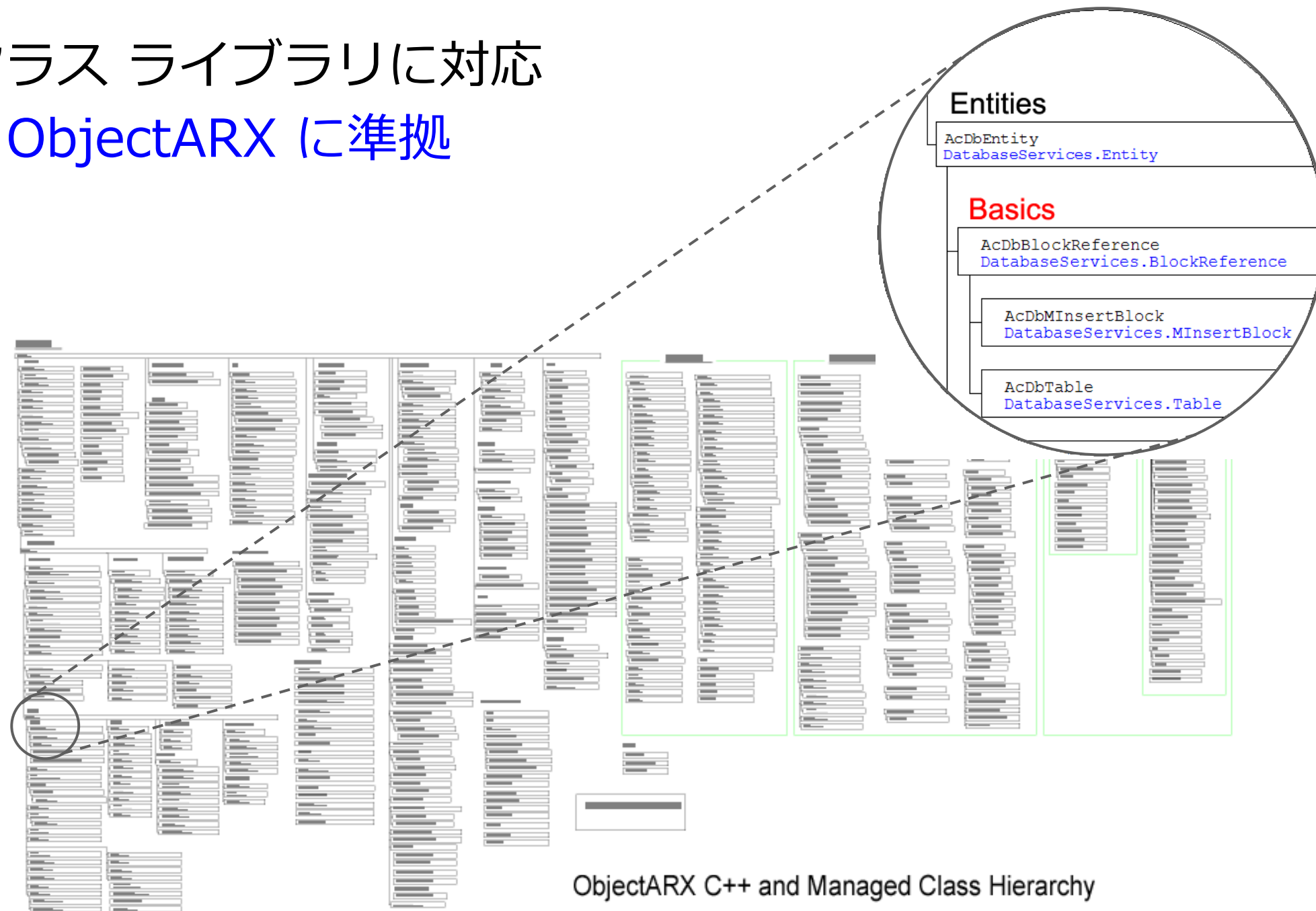
幾何演算関連

**Autodesk.AutoCAD.DatabaseServices**

グラフィカル、非グラフィカルオブジェクト等図面データベース関連

# AutoCAD マネージ クラス ライブラリ

- ObjectARX クラス ライブラリに対応
  - API 操作も ObjectARX に準拠



ObjectARX C++ and Managed Class Hierarchy

# .NET API プログラミングの利点

- ガベージコレクションによる自動メモリ管理
  - アンマネージ C++：潜在的なメモリ リークの危険性

```
char *pName=(char*)malloc(128);  
  
strcpy(pName,"Hello");  
  
//...  
  
free(pName);
```

- マネージ言語：ガベージ コレクションによりメモリを解放

```
String *pName=new String("Hello");           // C++  
  
Dim Name As String = "Hello"                 \ VB.NET  
  
String Name = "Hello";                        // C#
```

# AutoCAD の .NET 環境って？

- AutoCAD .NET API の仕組みの中では ...
  - COM ベースと .NET Framework ベースの 2 種類のプログラムがある
- COM を使ったプログラムの記述方法
  - VBA とほぼ同じ書式です（VB.NET の新しい仕様には注意）
  - VBA 資産の移行先として利用できます
- .NET Framework ライブラリを使ったプログラムの記述方法
  - .NET ならではの機能やサービスを利用できます



# 開発実習

# トレーニング コース内容

- AutoCAD のカスタマイズとは？
- .NET API の利点
- Lesson 1：プロジェクトとコマンドの作成
- Lesson 2：グラフィカル オブジェクトの作成
- Lesson 3：ユーザとの対話
- Lesson 4：オブジェクトの編集
- Lesson 5：パレット ダイアログの作成
- Lesson 6：非グラフィカル オブジェクトとの連携
- Lesson 7：他のアプリケーションとの連携
- Lesson 8：イベント ハンドリング
- Lesson 9：カスタムデータ

# Lesson 1

## プロジェクトとコマンドの作成

# Lesson

# AutoCAD .NET アプリケーション作成の用意

- AutoCAD 2022
  - アセンブリ ファイルで API を公開
  - [accoremgd.dll](#)、[acdbmgd.dll](#)、[acmgd.dll](#)、その他
  - .NET アプリケーションは AutoCAD にロードして実行
- Visual Studio 2019 Version 16 以降
  - .NET Framework 4.8 をサポートする Visual Studio
  - AutoCAD .NET API アセンブリを参照
  - ビルド（コンパイル）してバイナリ ファイル（DLL ファイル）を作成
- ObjectARX SDK
  - AutoCAD .NET API に関するドキュメント（英語）とサンプルを提供
  - 本来は ObjectARX API 用の Software Development Kit

# .NET API 関連のドキュメント

- AutoCAD 2022 オンラインヘルプ内
  - AutoCAD .NET 開発者用ガイド(オンライン、日本語)
    - <http://help.autodesk.com/view/OARX/2022/JPN/?guid=GUID-C3F3C736-40CF-44A0-9210-55F6A939B6F2>
- ObjectARX SDK 内
  - AutoCAD Managed .NET Developer's Guide(arxmgd\_dev.chm 、英語)
    - オンライン AutoCAD .NET 開発者用ガイドの英語版
  - AutoCAD Managed Class Reference(arxmgd.chm、英語)
    - 名前空間単位のリファレンス
    - メソッドとプロパティを ObjectARX 関数と対比

# Lesson 1: プロジェクトとコマンドの作成

- プロジェクトを作成して Hello コマンドを実装
- Hello コマンドでプロンプトに “HelloWorld!” を表示



- 目的
  - AutoCAD .NET アプリケーション用プロジェクトの作成
  - AutoCAD アセンブリの参照設定
  - コマンド登録と呼び出される関数の関係を理解
  - ロード方法と実行の確認



# Lesson 1: プロジェクトとコマンドの作成

## 1. Visual Studio 2019 で クラス ライブラリ プロジェクト作成

### 新しいプロジェクトの作成

最近使用したプロジェクト テンプレート(R)

VB プロジェクトの場合

テンプレートの検索(S) (Alt+S)

すべてクリア(C)

Visual Basic

Windows

すべてのプロジェクトの種類(...)



クラス ライブラリ (.NET Framework)

VB クラス ライブラリ (.dll) を作成するためのプロジェクトです

Visual Basic

Windows

ライブラリ



単体テスト プロジェクト (.NET Framework)

MSTest 単体テストを含むプロジェクト。

Visual Basic

Windows

テスト



Windows フォーム コントロール ライブラリ (.NET Framework)

Windows Forms (WinForms) アプリケーションで使用するコントロールを作成するためのプロジェクトです

Visual Basic

Windows

デスクトップ

ライブラリ



WPF ブラウザー アプリ (.NET Framework)

Windows Presentation Foundation ブラウザー アプリケーションです

Visual Basic

XAML

Windows

デスクトップ



WPF ユーザー コントロール ライブラリ (.NET Framework)

Windows Presentation Foundation ユーザー コントロール ライブラリです

Visual Basic

XAML

Windows

デスクトップ

ライブラリ

次へ(N)

### 新しいプロジェクトを構成します

クラス ライブラリ (.NET Framework)

Visual Basic

Windows

ライブラリ

プロジェクト名(N)

ClassLibrary1

場所(L)

C:\トレーニング - デスクトップ製品 API\AutoCAD 2022 .NET API Training¥

ソリューション名(M) ⓘ

ClassLibrary1

☐ ソリューションとプロジェクトを同じディレクトリに配置する(D)

フレームワーク(F)

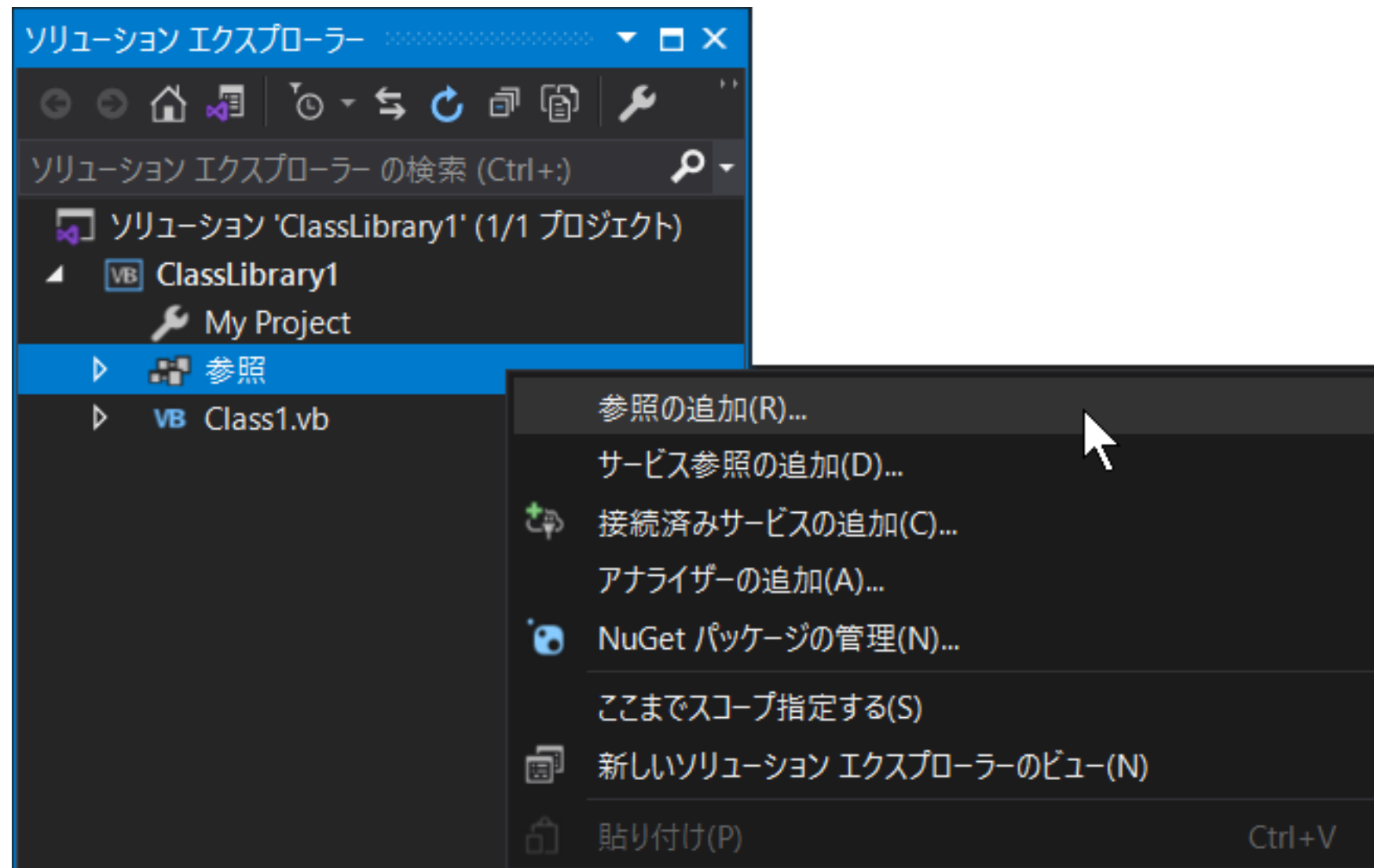
.NET Framework 4.8

戻る(B)

作成(C)

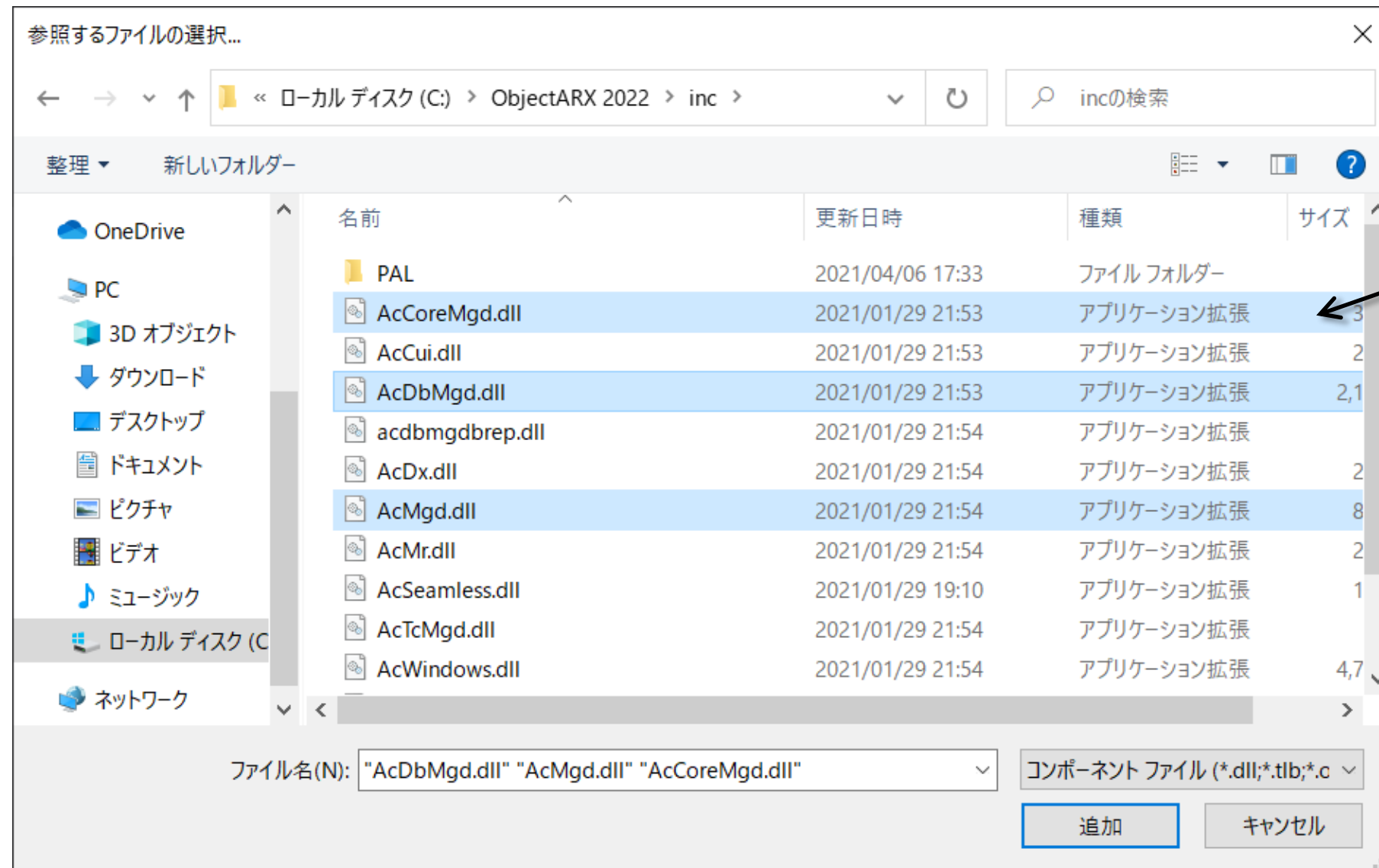
# Lesson 1: プロジェクトとコマンドの作成

## 2. ソリューション エクスプローラから [参照の追加] を選択



# Lesson 1: プロジェクトとコマンドの作成

## 3. accoremgd.dll、acdbmgd.dll と acmgd.dll を選択



参照設定後は[ローカルコピー] プロパティを True から False に設定してください

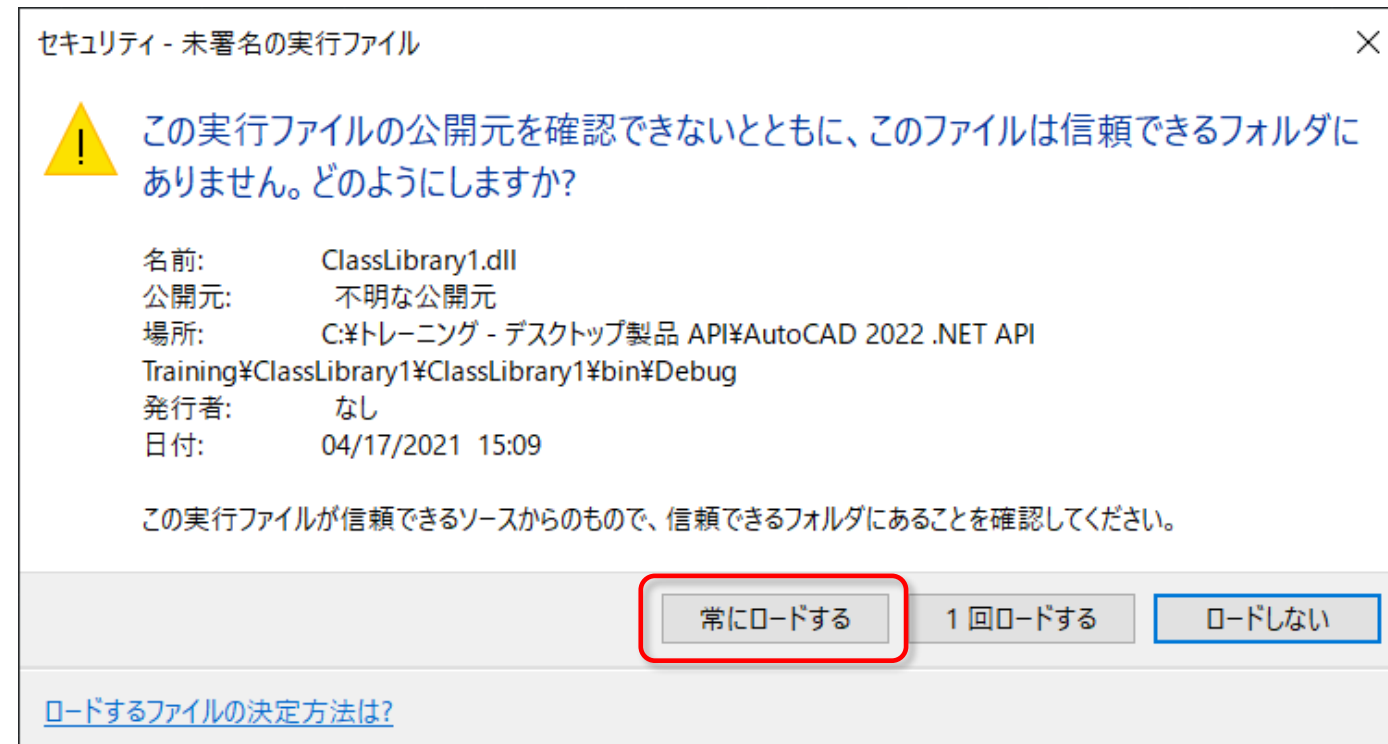
# Lesson 1: プロジェクトとコマンドの作成

## 4. グローバル スコープに名前空間を追加してクラスにコマンドを追加

```
Imports Autodesk.AutoCAD.ApplicationServices ← 名前空間のインポート
Imports Autodesk.AutoCAD.Runtime
Public Class Class1
    <CommandMethod("Hello")> ← コマンド定義記述とコマンド関数
    Public Sub MyCommand()
        Application.DocumentManager.MdiActiveDocument.Editor.WriteMessage( _
            vbCrLf + "Hello World!")
    End Sub
End Class
```

# Lesson 1: プロジェクトとコマンドの作成

## 5. コンパイル後に NETLOAD コマンドでアセンブリをロード



## 6. HELLO コマンドを実行



# アプリケーションとコマンドの関係

## ■ アプリケーションは通常クラス単位（必須ではありません）

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Runtime
Public Class Class1
    <CommandMethod("Hello")> _
    Public Shared Sub MyCommand()
        Application.DocumentManager.MdiActiveDocument.Editor.WriteMessage( _
            vbCrLf + "Hello World!")
    End Sub
End Class
```

Public Class XXX ~ End Class の間に複数の  
コマンド宣言 <CommandMethod> を記述

## ■ コマンドの登録は宣言で実施

```
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Runtime
Public Class Class1
    <CommandMethod("Hello")> _
    Public Shared Sub MyCommand()
        Application.DocumentManager.MdiActiveDocument.Editor.WriteMessage( _
            vbCrLf + "Hello World!")
    End Sub
End Class
```

<CommandMethod> で指定したコマンド名 "Hello" で  
接続する Sub プロシージャが呼び出す、という "宣言"



# コマンドの種類 – コマンド フラグ

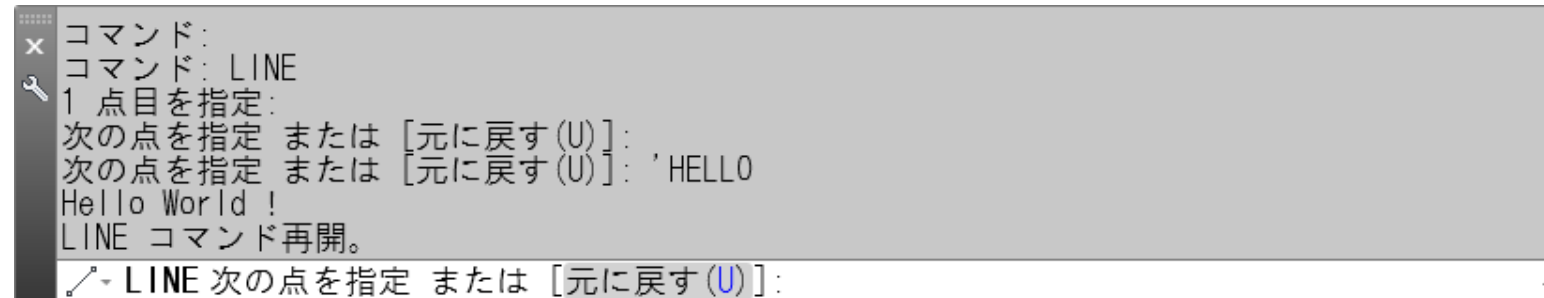
- コマンドの振る舞いを指定する
- 優先コマンド フラグ（いずれかを指定）
  - `CommandFlags.Modal` : 通常のコマンド
  - `CommandFlags.Transparent` : 割り込みコマンド
- 優先コマンド フラグとの組み合わせ可能な代表的なフラグ
  - `CommandFlags.NoPaperSpace` : モデル空間のみで実行可能なコマンド
  - `CommandFlags.NoTileMode` : ペーパー空間でのみ実行可能なコマンド
  - `CommandFlags.NoPerspective` : 平行投影でのみ実行可能なコマンド
  - `CommandFlags.UsePickSet` : 先行選択が可能なコマンド
  - `CommandFlags.Redraw` : 先行選択をクリアするコマンド

# コマンドの種類 – コマンド フラグ

## ■ Transparent と NoPaperSpace フラグを指定した Hello コマンド

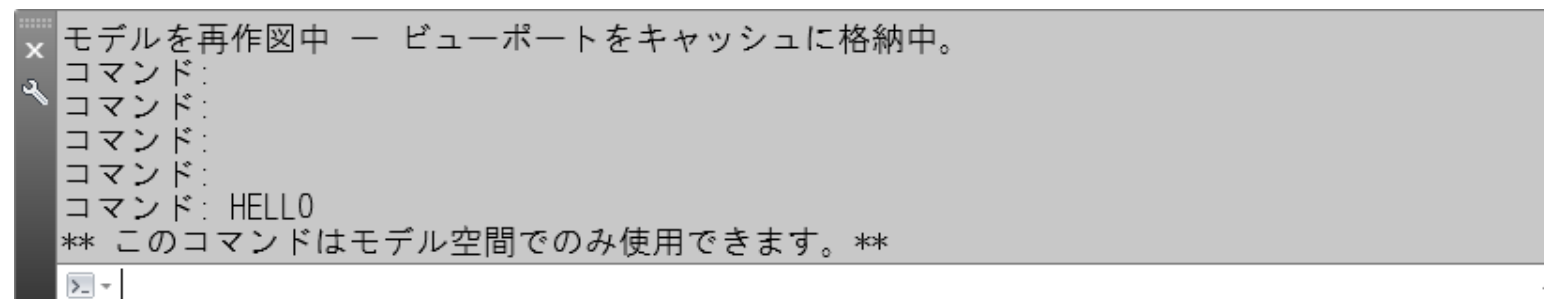
```
<CommandMethod("Hello", CommandFlags.Transparent Or CommandFlags.NoPaperSpace)> _  
Public Shared Sub Test()  
    Application.DocumentManager.MdiActiveDocument.Editor.WriteMessage( _  
                                                vbCrLf + "Hello World!")  
End Sub
```

## ■ 割り込みが可能



コマンド:  
コマンド: LINE  
1 点目を指定:  
次の点を指定 または [元に戻す(U)]:  
次の点を指定 または [元に戻す(U)]: 'HELLO  
Hello World!  
LINE コマンド再開。  
LINE 次の点を指定 または [元に戻す(U)]:

## ■ 優先コマンド フラグとの組み合わせ可能なフラグ



モデルを再作図中 – ビューポートをキャッシュに格納中。  
コマンド:  
コマンド:  
コマンド:  
コマンド:  
コマンド: HELLO  
\*\* このコマンドはモデル空間でのみ使用できます。 \*\*

# コマンドの種類 – コマンド グループ

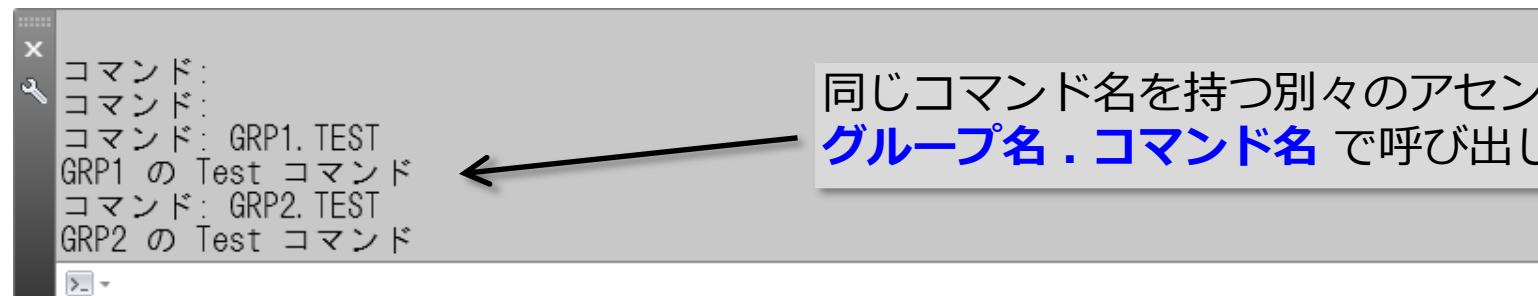
- 異なるアセンブリの同一コマンド名を区別
- ABC.dll アセンブリ内の Test コマンド = GRP1 グループ

通常、グループ名は英数字 4 文字で指定

```
<CommandMethod("GRP1", "Test" , CommandFlags.Modal)> _  
Public Shared Sub Test()  
    Application.DocumentManager.MdiActiveDocument.Editor.WriteMessage(vbCrLf + "GRP1 の Test コマンド")  
End Sub
```

- XYZ.dll アセンブリ内の Test コマンド = GRP2 グループ

```
<CommandMethod("GRP2", "Test" , CommandFlags.Modal)> _  
Public Shared Sub Test()  
    Application.DocumentManager.MdiActiveDocument.Editor.WriteMessage(vbCrLf + "GRP2 の Test コマンド")  
End Sub
```



同じコマンド名を持つ別々のアセンブリが同時にロードされた状態では  
グループ名 . コマンド名 で呼び出したいコマンドを識別できる！

# .NET API アプリケーションのロード制御

- dll ファイルのロード制御
  - AutoCAD 標準コマンドを使ったロード
    - NETLOAD コマンド
  - システム レジストリを使ったロード制御
    - **デマンド ロード**
  - 自動起動ファイルを使ったロード制御
    - acad.lsp、acaddoc.lsp
  - **オートローダー（自動ローダー）** を使ったロード制御
    - XML パッケージ ファイルにロード条件を記述
- .NET API アセンブリは原則ロード解除の不可



# Lesson 2

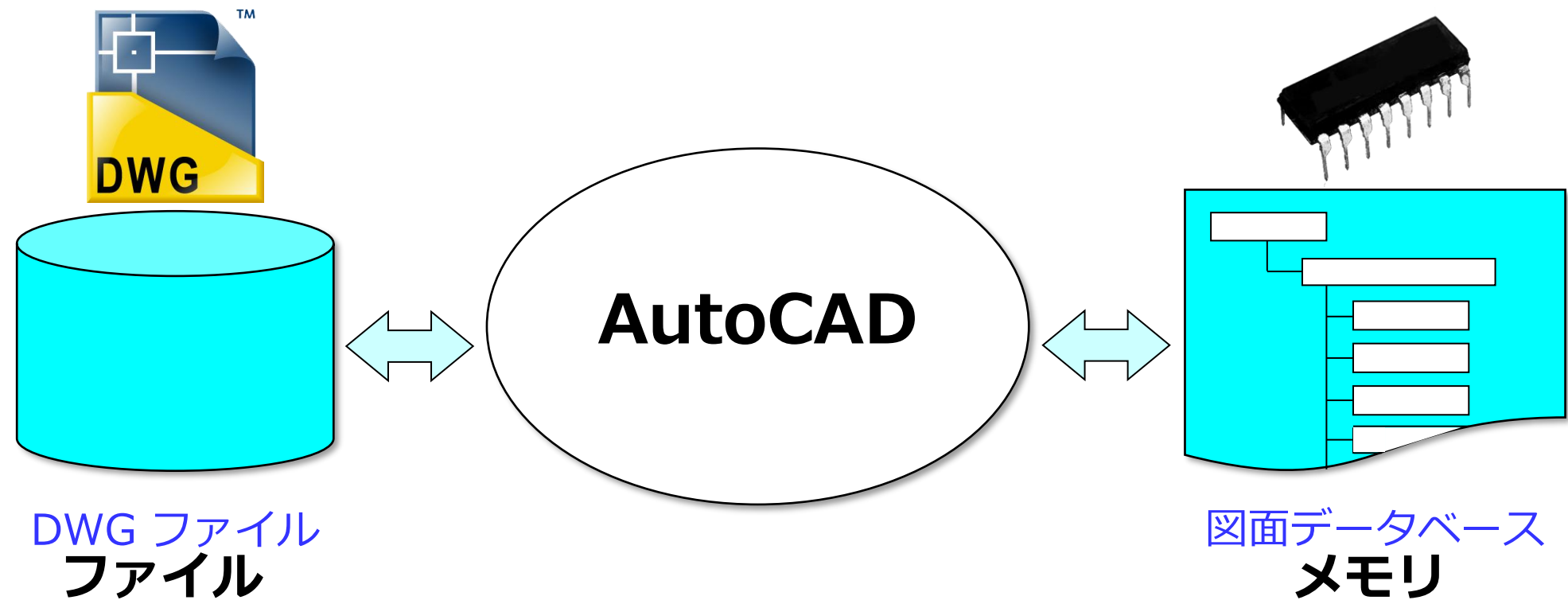
## グラフィカル オブジェクトの作成

# Lesson

# AutoCAD オブジェクトの種類

- AutoCAD 上に図面を表示するということは …

- DWG ファイルをメモリに展開すること

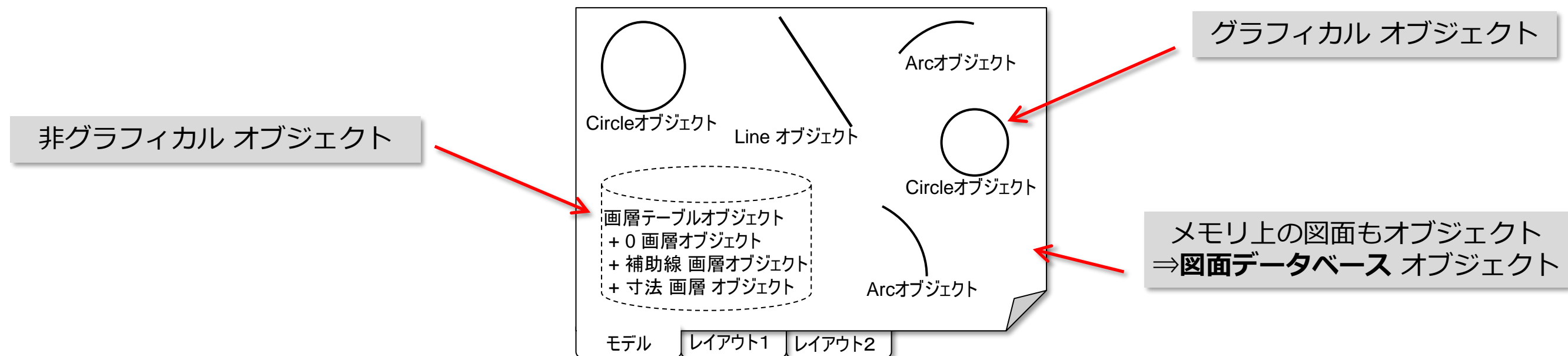


- メモリ上の図面に存在しているのは …



# AutoCAD オブジェクトの種類

- グラフィカル オブジェクト
  - 目に見える図面上のエンティティ
  - 線分、円、円弧、ポリライン、楕円、ソリッド...
- 非グラフィカル オブジェクト
  - 目に見えない情報オブジェクト
  - 画層、線種、寸法スタイル、文字スタイル...



# AutoCAD .NET API でオブジェクトを扱う

- 各オブジェクトは特性別に クラス で定義される
  - 例)
    - 線分オブジェクト : Line クラス
    - 円オブジェクト : Circle クラス
    - 図面データベース オブジェクト : Database クラス
    - 画層テーブル オブジェクト : LayerTable クラス
    - “0” 画層オブジェクト : LayerTableRecord クラス
    - ブロック テーブル オブジェクト : BlockTable クラス
    - モデル空間 オブジェクト : BlockTableRecord クラス
    - “TEST” ブロック定義オブジェクト : BlockTableRecord クラス
    - ...
  - 図面データベース内の “オブジェクト”
    - メモリ上に展開されたクラスのインスタンス (複製)

# 言葉の整理

## ■ オブジェクト

- メモリ上に展開された図面に含まれる、目に見える図形や目に見えないスタイルなど、個々の実体を指す
- ※ モデル空間上の赤い円と青い円は別オブジェクト

## ■ クラス

- 公開されたライブラリに含まれる、個々のオブジェクトの定義（振る舞い = メソッド、プロパティ）
- ※ 赤い円も青い円も Circle クラスで定義されている

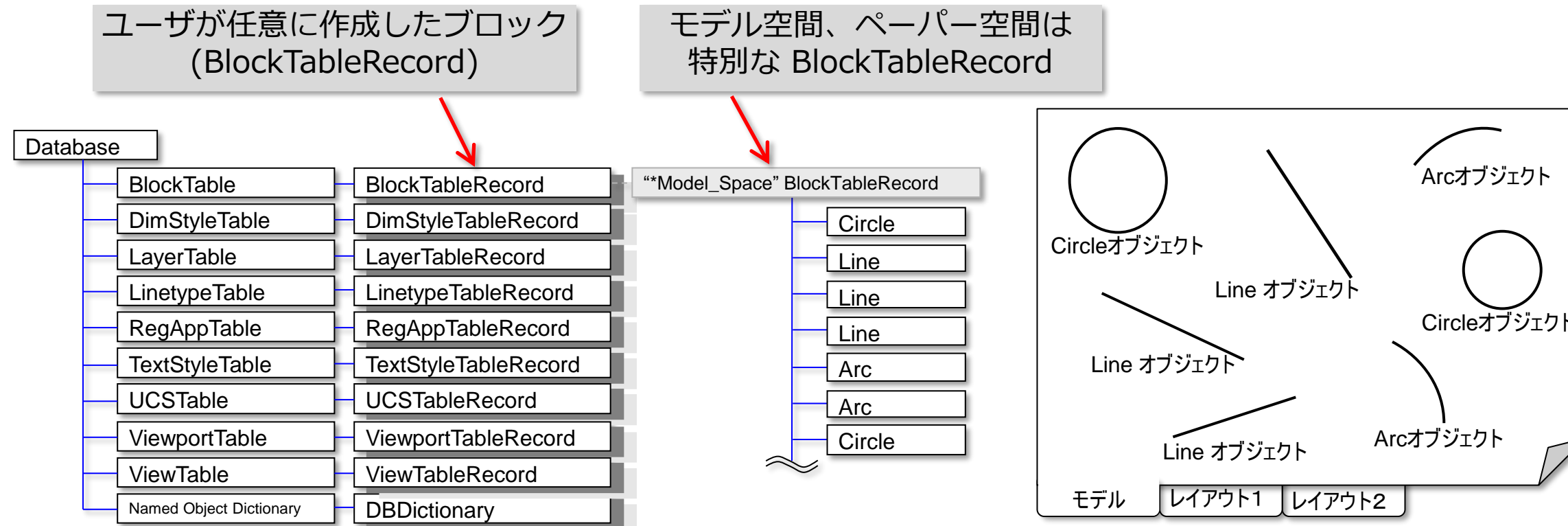
```
Dim oLine1 As Line  
Dim oLine2 As Line
```



実際のプログラムでも **クラス** と **オブジェクト** の違いを明確に意識しなければなりません !!

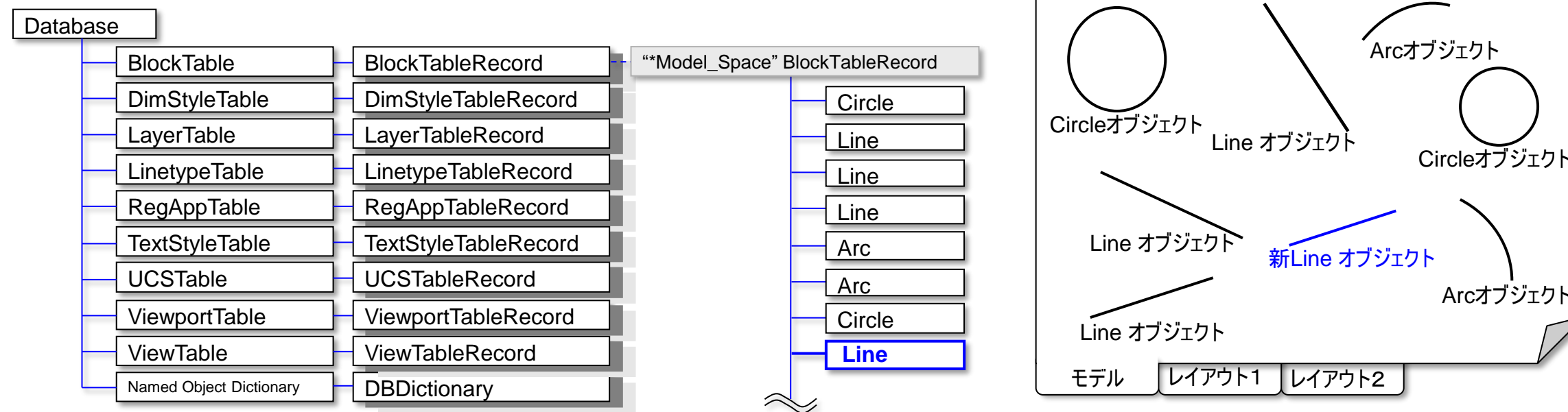
# メモリ上の図面構造（図面データベース）

- オブジェクトが階層的に整理されている
  - トップ オブジェクトが 図面データベース オブジェクト
  - オーナシップ接続 によって論理的な つながり を保持
  - オブジェクトは然るべき コンテナ オブジェクト に格納



# メモリ上の図面構造に線分を追加する

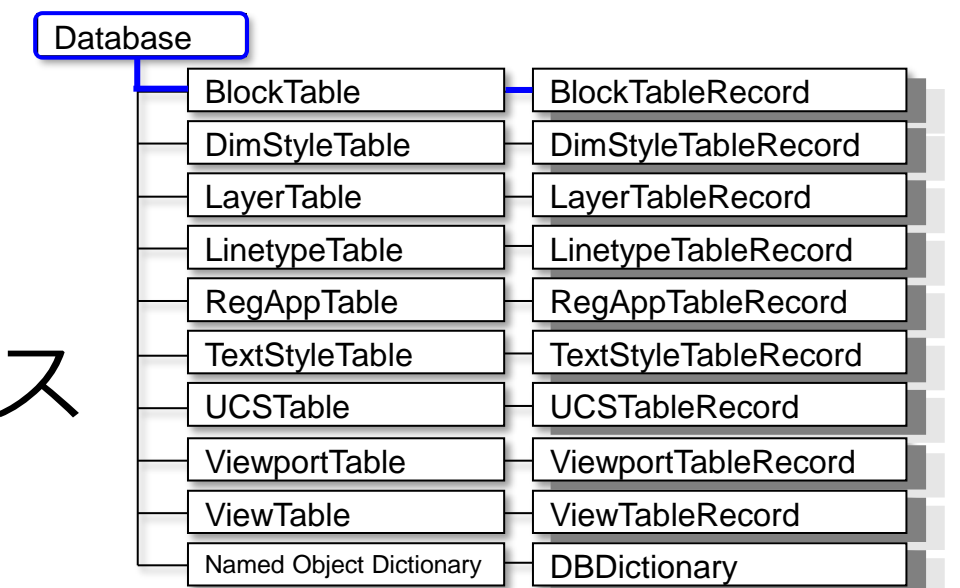
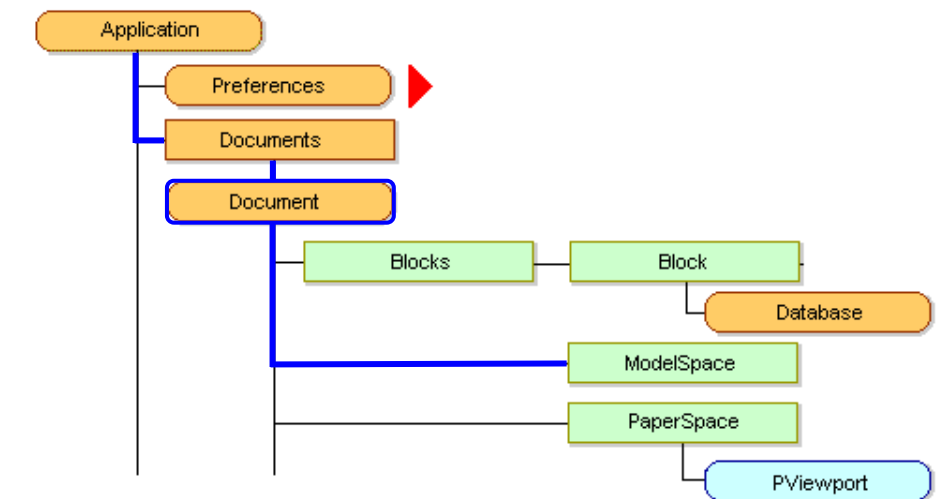
- 新しく Line クラスを元にオブジェクトを作成して
- モデル空間を表す BlockTableRecord に論理上の つながり をつける
  - オーナーシップ接続の確立



- BlockTableRecord オブジェクトを取得する必要あり

# メモリ上のオブジェクト階層の追跡

- COM でのオブジェクト モデル上での追跡
  - Application オブジェクトありき
  - Application オブジェクトから各プロパティを介して下位オブジェクトにアクセス
  - ThisDrawing キーワード
    - 現在の Document オブジェクト
- .NET API は図面データベースを追跡
  - Database オブジェクトありき
  - 下位オブジェクトへは **オブジェクトID** 介してアクセス
  - 現在の図面データベースを取得
    - Database オブジェクト



# AutoCAD API で利用するオブジェクト識別子

- エンティティ名
  - AutoLISP、ObjectARX (旧ADS) で使用
  - セッション毎に更新され図面には保存されない
  - 複数図面オープン的环境下での使用は考慮されていない
- ハンドル番号
  - API での利用は必須ではないが全 API で取得可能
  - セッション毎に不変であり図面に保存される
  - 16バイトの文字列で表現される
  - MDI環境では重複の可能性あり
- オブジェクト ID
  - ObjectARX、[.NET API](#) で使用
  - セッション毎に更新され図面には保存されない
  - ObjectId クラスで表現される
  - 複数図面オープン的环境下での使用が考慮されている



# 現在の図面データベースの取得

- 図面データベース
  - Database クラスで定義
  - AutoCAD上は複数の図面をオープン可能
  - インスタンス Database で識別
- カレント図面データベース
  - AutoCAD上でフォーカスを持つ MDI 子ウィンドウ
  - プログラムによって変わる場合もある
- 現在編集対象となっている図面データベース
- `HostApplicationServices.WorkingDatabase` プロパティ
- 現在の図面データベースの `Database` オブジェクト を返す

# Database からの下位オブジェクト追跡

- Database のプロパティは直接オブジェクト取得不可
  - 下位オブジェクトのオブジェクト ID を取得
- 下位オブジェクト ID 取得プロパティ例
  - Database.BlockTableId プロパティ
    - ブロックテーブルのオブジェクトIDを返す
  - Database DimStyleTableId プロパティ
    - 寸法スタイルテーブルのオブジェクトIDを返す
  - Database.LayerTableId プロパティ
    - 画層テーブルのオブジェクトIDを返す
  - Database.LinetypeTableId プロパティ
    - 線種テーブルのオブジェクトIDを返す
  - Database.TextStyleTableId プロパティ
    - 字体スタイルテーブルのオブジェクトIDを返す

# オブジェクト操作時の原則

1. オブジェクト ID でオブジェクトを特定
2. オブジェクト ID でオブジェクトを **オープン**
  - ここではじめてオブジェクトを取得
3. オブジェクトに **アクション**
  - メソッド、プロパティにアクセス
4. オブジェクトを **クローズ**
  - ここでオブジェクトへのアクションがコミットされる

# モデル空間に線分を追加する .NET API 例

```
<CommandMethod("CreateLine")> _  
Public Shared Sub CreateLine()  
  
    Dim oDb As Autodesk.AutoCAD.DatabaseServices.Database = _  
        Autodesk.AutoCAD.DatabaseServices.HostApplicationServices.WorkingDatabase  
    Dim btObjId As Autodesk.AutoCAD.DatabaseServices.ObjectId = oDb.BlockTableId  
    Dim oBt As Autodesk.AutoCAD.DatabaseServices.BlockTable = _  
        btObjId.Open(OpenMode.ForRead)  
    Dim btrObjId As Autodesk.AutoCAD.DatabaseServices.ObjectId = _  
        oBt.Item("*MODEL_SPACE")  
    Dim oBtr As Autodesk.AutoCAD.DatabaseServices.BlockTableRecord = _  
        btrObjId.Open(OpenMode.ForWrite)  
    Dim oLine As Autodesk.AutoCAD.DatabaseServices.Line = _  
        New Autodesk.AutoCAD.DatabaseServices.Line(ptStart, ptEnd)  
    oBtr.AppendEntity(oLine)  
    oBtr.Close()  
    oLine.Close()  
  
End Sub
```

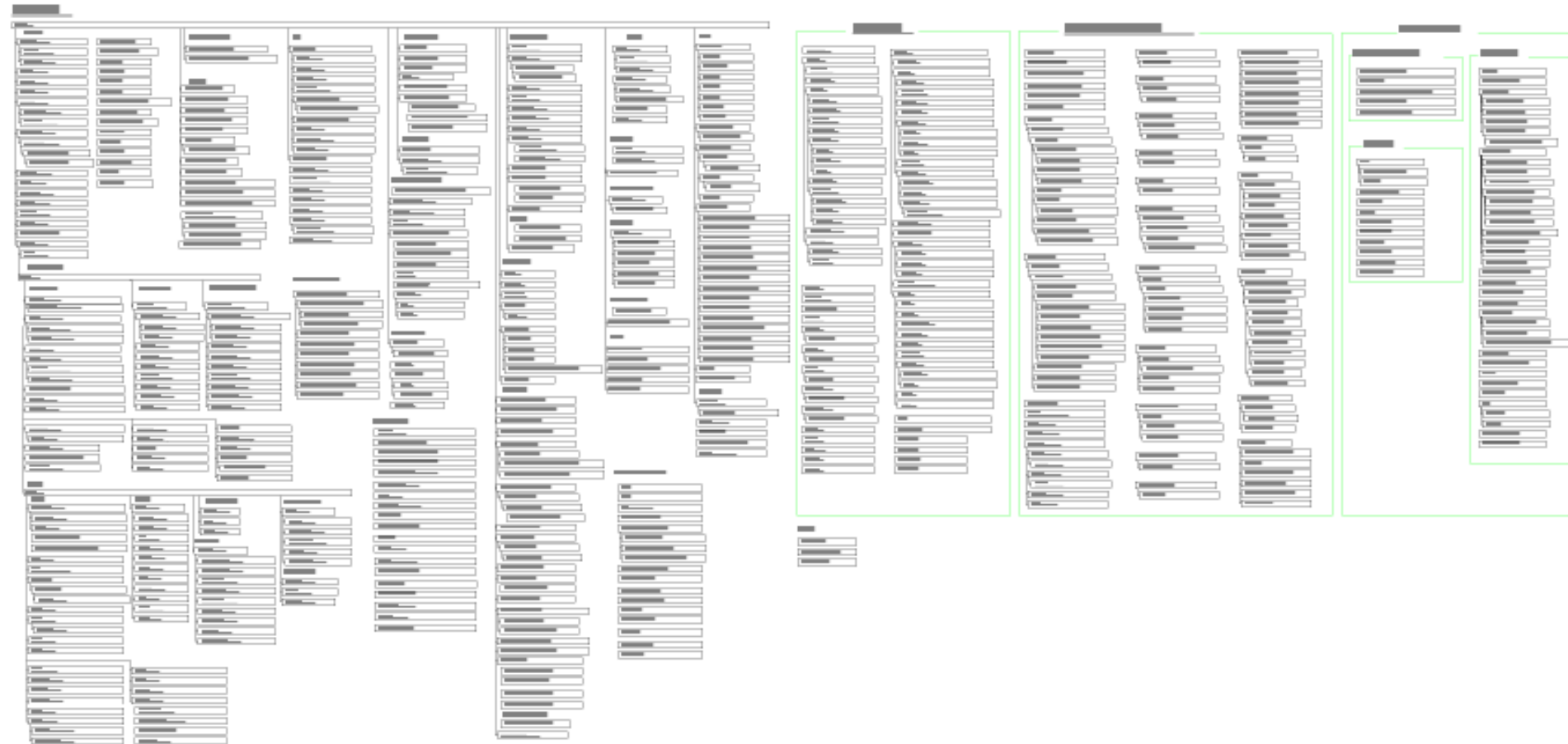
オブジェクト呼び出しが長くて煩雑過ぎる !!

```
Dim ptStart As Autodesk.AutoCAD.Geometry.Point3d = _  
    New Autodesk.AutoCAD.Geometry.Point3d(0.0, 0.0, 0.0)  
Dim ptEnd As Autodesk.AutoCAD.Geometry.Point3d = _  
    New Autodesk.AutoCAD.Geometry.Point3d(100.0, 100.0, 0.0)  
Dim oLine As Autodesk.AutoCAD.DatabaseServices.Line = _  
    New Autodesk.AutoCAD.DatabaseServices.Line(ptStart, ptEnd)  
oBtr.AppendEntity(oLine)  
oBtr.Close()  
oLine.Close()
```

End Sub

# 名前空間

- AutoCAD .NET API アセンブリが公開しているクラス群
- 大規模な階層構造を持つ



- 名前空間：公開クラスを体系化して簡単に呼び出す機構

# 名前空間を使う

- メソッドやプロパティ呼び出しが長く煩雑になる
  - 例) 3次元点変数の宣言とオブジェクトの作成

```
Dim oPt As Autodesk.AutoCAD.Geometry.Point3d  
oPt = New Autodesk.AutoCAD.Geometry.Point3d(0.0, 0.0, 0.0)
```

- Imports キーワードを使って名前空間の指定を省略する

```
Imports Autodesk.AutoCAD.Geometry  
Dim oPt As Point3d  
oPt = New Point3d(0.0, 0.0, 0.0)
```

- ✓ Imports はファイルの先頭で宣言する
- ✓ 複数行の Imports で異なる名前空間をインポート可能

- 変数に代入して呼び出す

```
Dim oGe As Autodesk.AutoCAD.Geometry  
Dim oPt As oGe.Point3d  
oPt = New oGe.Point3d(0.0, 0.0, 0.0)
```

# AutoCAD .NET API の主な名前空間

- [Autodesk.AutoCAD.Runtime](#)
  - 例外処理など基本機能関連
- [Autodesk.AutoCAD.ApplicationServices](#)
  - AutoCAD フレーム ウィンドウやキュメントアクセス関連
- [Autodesk.AutoCAD.DatabaseServices](#)
  - グラフィカル、非グラフィカル オブジェクト等図面データベース関連
- [Autodesk.AutoCAD.Geometry](#)
  - 幾何オブジェクトや幾何演算関連
- [Autodesk.AutoCAD.EditorInput](#)
  - 選択、入力等ユーザ対話関連
- [Autodesk.AutoCAD.PlottingServices](#)
  - 印刷サービス関連
- [Autodesk.AutoCAD.Windows](#)
- ユーザ インタフェース関連
- [Autodesk.AutoCAD.GraphicsInterface](#)
  - ベクトル描画機構関連



# モデル空間に線分を追加 … : 名前空間を使うと

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateLine")> _
Public Shared Sub CreateLine()

    Dim oDb As Database = HostApplicationServices.WorkingDatabase
    Dim btObjId As ObjectId = oDb.BlockTableId
    Dim oBt As BlockTable = btObjId.Open(OpenMode.ForRead)
    Dim btrObjId As ObjectId = oBt.Item("*MODEL_SPACE")
    oBt.Close()
    Dim oBtr As BlockTableRecord = btrObjId.Open(OpenMode.ForWrite)
    Dim ptStart As Point3d = New Point3d(0.0, 0.0, 0.0)
    Dim ptEnd As Point3d = New Point3d(100.0, 100.0, 0.0)
    Dim oLine As Line = New Line(ptStart, ptEnd)
    oBtr.AppendEntity(oLine)
    oBtr.Close()
    oLine.Close()

End Sub
```

# コードの補足 – その1

- 点 もオブジェクトとして扱う
  - Point3d オブジェクト
- オブジェクトの追加は 2 つの手順で実施
  1. オブジェクトは New で新規作成が原則
  2. その後、オーナーシップ接続手続き実施
- AppendEntity メソッド
  - グラフィカル オブジェクトを追加
  - BlockTableRecord にグラフィカル オブジェクトをつなげます
  - このメソッド呼び出しでオーナーシップ接続が確立
- オーナーシップ接続
  - オーナーシップ接続を持つオブジェクトのみが DWG ファイル保存対象に
  - オーナーシップ接続のあるオブジェクトはクローズは必須

# コードの補足 – その2

- Item メソッドで指定
  - 特別なブロックテーブルレコード名
  - “\*MODEL\_SPACE” ⇒ モデル空間
  - “\*PAPER\_SPACE” ⇒ ペーパー空間 (レイアウト1)
  - “\*PAPER\_SPACE0” ⇒ ペーパー空間 (レイアウト2)
  - …以降数字の増加
- ブロックテーブルレコードはブロック定義を示します
  - 上記以外は通常のブロック定義
  - ユーザ作成時の任意名で取得可能

# コードの補足 – その3

- オブジェクトのオープンにはオープンモードが必要
  - `OpenMode.ForWrite` : 書き込みオープン
  - `OpenMode.ForRead` : 読み込みオープン
  - `OpenMode.ForNotify` : 通知オープン

```
        :  
Dim oBtr As BlockTableRecord = btrObjId.Open(OpenMode.ForWrite)  
        :  
oBtr.AppendEntity(oLine)  
oBtr.Close()  
        :
```

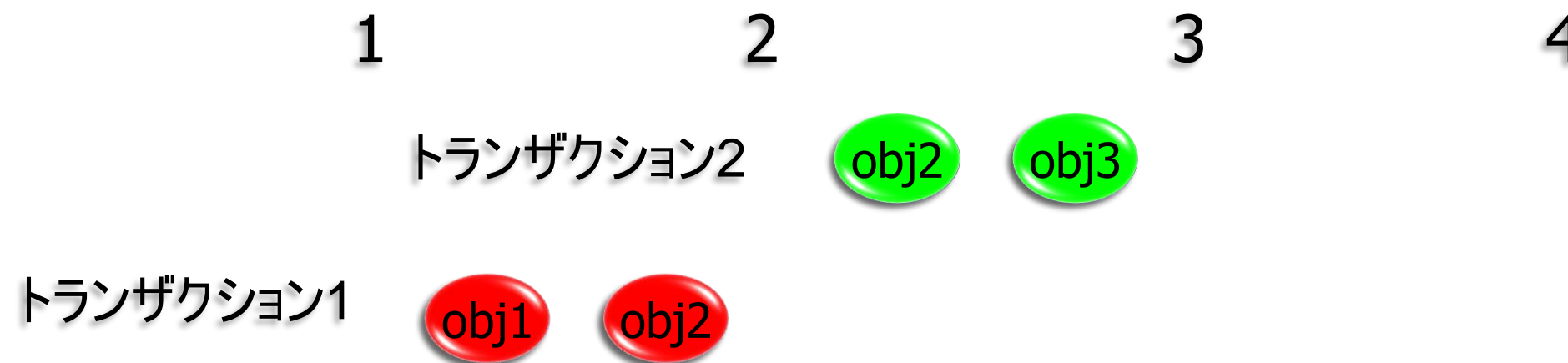


モデル空間 BlockTableRecord に Line 追加するので  
BlockTableRecord は 書き込みモード でオープン

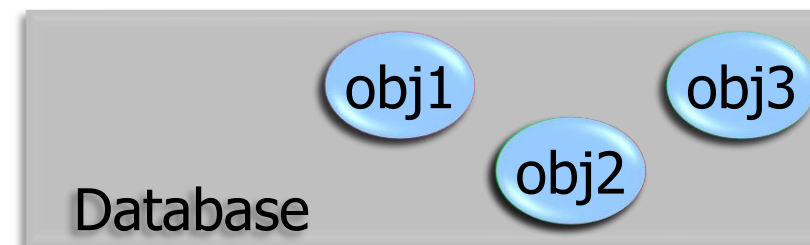
- 読み込みモードで書き込み操作をするとクラッシュします !!
- オープン、アクション、クローズの確認
  - アクション後にはクローズが必須 !!
  - クローズし忘ればクラッシュにつながります !!

# トランザクションの利用

- オープン/クローズ メカニズムに替わる機構
  - 一連のオブジェクト操作を単位化
  - ネストしたトランザクションを正しく管理



- 利点
  - Close メソッドの呼び出しなし
  - コミット操作を一括処理



# トランザクションの利用

- トランザクション マネージャ
  - TransactionManager クラスで定義
  - トランザクションの開始、終了、中断を管理
- トランザクションの開始
  - `TransactionManager.StartTransaction` メソッド
  - Transaction オブジェクト を返す
- トランザクションの終了
  - 処理確定：コミットという
  - `Transaction.Commit` メソッド
- トランザクションの中断
  - 処理のキャンセル（開始前の状態に戻す）：ロールバックという
  - `Transaction.Abort` メソッド

# トランザクションの利用

1. トランザクション中のオブジェクト取得
  - `Transaction.GetObject` メソッド
  - オープンする オブジェクトID と オープンモード を指定
2. トランザクション中のオブジェクト追加の記録
  - `BlockTableRecord.AppendEntity` はオーナーシップを接続
  - このオブジェクト追加のトランザクションへの記録は必須！
  - `Transaction.AddNewlyCreatedDBObject` メソッド
3. トランザクション使用後の後処理
  - `Transaction.Dispose` メソッド を必ず呼び出す！

**トランザクションは通常エラー処理とともに使用！… 詳細は後のセクションで説明**



# モデル空間に線分を追加… : トランザクションを利用

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.Geometry

<CommandMethod("CreateLine")> _
Public Shared Sub CreateLine()
    Dim oDb As Database = HostApplicationServices.WorkingDatabase
    Dim oTr As Transaction = oDb.TransactionManager.StartTransaction
    Dim ptStart As Point3d = New Point3d(0.0, 0.0, 0.0)
    Dim ptEnd As Point3d = New Point3d(100.0, 100.0, 0.0)
    Dim oLine As Line = New Line(ptStart, ptEnd)
    Dim oBt As BlockTable = oTr.GetObject(oDb.BlockTableId, OpenMode.ForRead)
    Dim oBtr As BlockTableRecord = _
        oTr.GetObject(oBt.Item("*MODEL_SPACE"), OpenMode.ForWrite)
    oBtr.AppendEntity(oLine)
    oTr.AddNewlyCreatedDBObject(oLine, True)
    oTr.Commit()
    oTr.Dispose()
End Sub
```

**.NET API ではトランザクションが標準操作です！**

# コードの補足

- オーナーシップ接続されたオブジェクトのクローズがない
  - トランザクションが暗黙のうちクローズを実施しています
- VB.NET の言語仕様
  - 宣言と同時に値を代入可能
  - オブジェクトの作成と同時に値の設定が可能
    - **コンストラクタ メソッド** という機能
    - 自動的に作成しているクラスの New メソッドを呼び出す
  - 同じ名前で異なる引き数（パラメータ）を持つメソッド
    - **オーバーロード** という機能

```
Dim ptStart As Point3d = New Point3d(  
    ▲1 / 3 ▼ New (plane As Autodesk.AutoCAD.Geometry.PlanarEntity, point As Autodesk.AutoCAD.Geometry.Point2d)  
  
Dim ptStart As Point3d = New Point3d(  
    ▲2 / 3 ▼ New (xyz0 As Double)  
  
Dim ptStart As Point3d = New Point3d(  
    ▲3 / 3 ▼ New (x As Double, y As Double, z As Double)
```

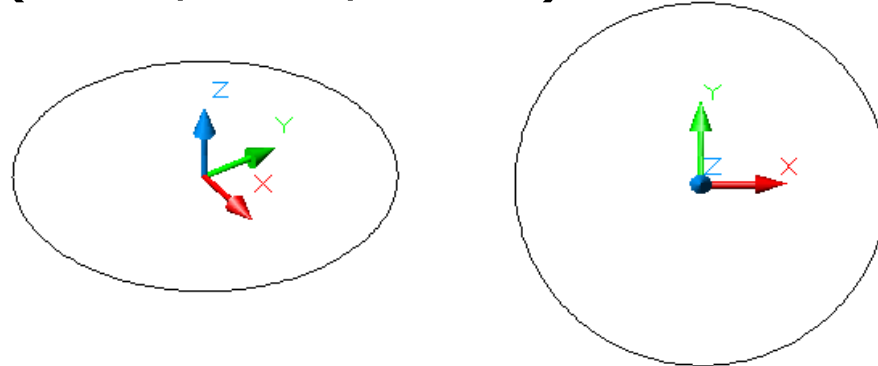
# Lesson 2: モデル空間に円を追加

- 円オブジェクト

- Circle クラスで定義

- 必要なパラメータ

- 中心点 ( 50.0, 50.0, 0.0 ) : Point3d オブジェクト
    - 半径 50.0 : Double オブジェクト
    - 法線ベクトル ( 0.0, 0.0, 1.0 ) : Vector3d オブジェクト



## 法線ベクトル

円の向いている方向をベクトルで表現するための情報。  
(0.0, 0.0, 1.0) は、Z軸方向と同じことを示します。  
XY平面では真円に見えます。

- トランザクション マネージャを使って追加

- 実装コマンド

- CreateCircle コマンド

## Lesson 2: モデル空間に円を追加

1. 現在の Database の取得
2. Database から Transaction を開始
3. Transaction から BlockTable をオープン
4. BlockTable から BlockTableRecord (モデル空間) を取得
5. Circle の要素 (Point3d、Vector3d、Double) を作成
6. 要素を元に Circle を作成
7. Transaction から BlockTableRecord をオープン
8. BlockTableRecord に Circle を追加
9. Transaction に追加した Circle を登録
10. Transaction を確定

# Lesson 3

## ユーザとの対話

# Lesson

# ユーザとの対話

- 様々な値をユーザ入力によって取得
  - 整数、実数、文字列、点 ...
- グラフィカル オブジェクトの選択
  - 単一オブジェクトの選択
  - 複数オブジェクトの選択（選択セット）
- ダイナミック入力

Autodesk.AutoCAD.EditorInput.Editor 名前空間を利用

# 整数値のユーザ入力

- 整数入力メソッド：GetInteger メソッド
  - プロンプト文字による オーバーロード あり
  - プロンプト文字用オブジェクト：PromptIntegerOptions
  - 戻り値オブジェクト：PromptIntegerResult
    - 入力値は Value プロパティ に Integer オブジェクトを格納
    - 入力判定は Status プロパティ に格納

```
Dim oEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

Dim oInt1 As PromptIntegerResult = oEd.GetInteger(vbCrLf & "整数を入力:")
If oInt1.Status <> PromptStatus.OK Then
    Exit Sub
End If
oEd.WriteMessage(vbCrLf + "入力値は " + oInt1.Value.ToString)
```

# 実数値のユーザ入力

- 実数入力メソッド：GetDouble メソッド
  - プロンプト文字による オーバーロード あり
  - プロンプト文字用オブジェクト：PromptDoubleOptions
  - 戻り値オブジェクト：PromptDoubleResult
    - 入力値は Value プロパティ に Double オブジェクトを格納
    - 入力判定は Status プロパティ に格納

```
Dim oEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

Dim oDb11 As PromptDoubleResult = oEd.GetDouble(vbCrLf & "実数を入力:")
If oDb11.Status <> PromptStatus.OK Then
    Exit Sub
End If
oEd.WriteMessage(vbCrLf + "入力値は " + oDb11.Value.ToString)
```



# 座標のユーザ入力

- 座標入力メソッド：GetPoint メソッド
  - プロンプト文字による オーバーロード あり
  - プロンプト文字用オブジェクト：PromptPointOptions
  - 戻り値オブジェクト：PromptPointResult
    - 入力値は Value プロパティ に Point3d オブジェクトを格納
    - 入力判定は Status プロパティ に格納

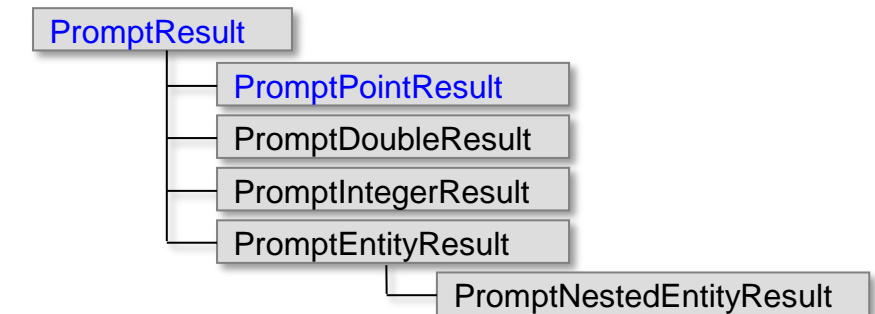
```
Dim oEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

Dim oPt1 As PromptPointResult = oEd.GetPoint(vbCrLf & "点を指定:")
If oPt1.Status <> PromptStatus.OK Then
    Exit Sub
End If

Dim oPtPmt As PromptPointOptions = New PromptPointOptions(vbCrLf & "点を指定:")
oPtPmt.UseBasePoint = True
oPtPmt.BasePoint = oPt1.Value
Dim oPt2 As PromptPointResult = oEd.GetPoint(oPtPmt)
If oPt2.Status <> PromptStatus.OK Then
    Exit Sub
End If
oEd.WriteMessage(vbCrLf + oPt1.Value.ToString + " - " + oPt2.Value.ToString)
```

# クラス継承

- クラス ライブラリ
  - クラスは目的や機能別に系統化されている
- 名前空間
  - クラスは 継承 機能を持っている
  - 子クラスは親クラスの機能を継承している
  - 例) Status プロパティは PromptPointResult クラスにはないが親クラスである PromptResult クラスにあるので利用可能

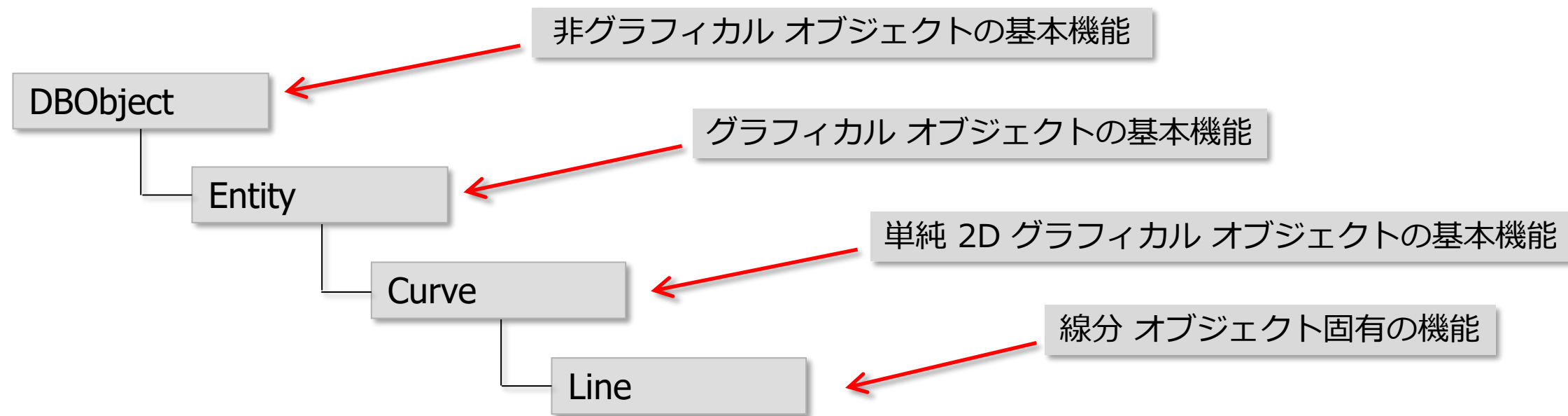


```
Dim oPt1 As PromptPointResult = oEd.GetPoint(vbCrLf & "点を指定:")
If oPt1.Status <> PromptStatus.OK Then
    Exit Sub
End If
```

オブジェクトブラウザや Managed Class Reference に記述がなくても親クラスのメソッドや、プロパティを利用することができます。

# クラス継承

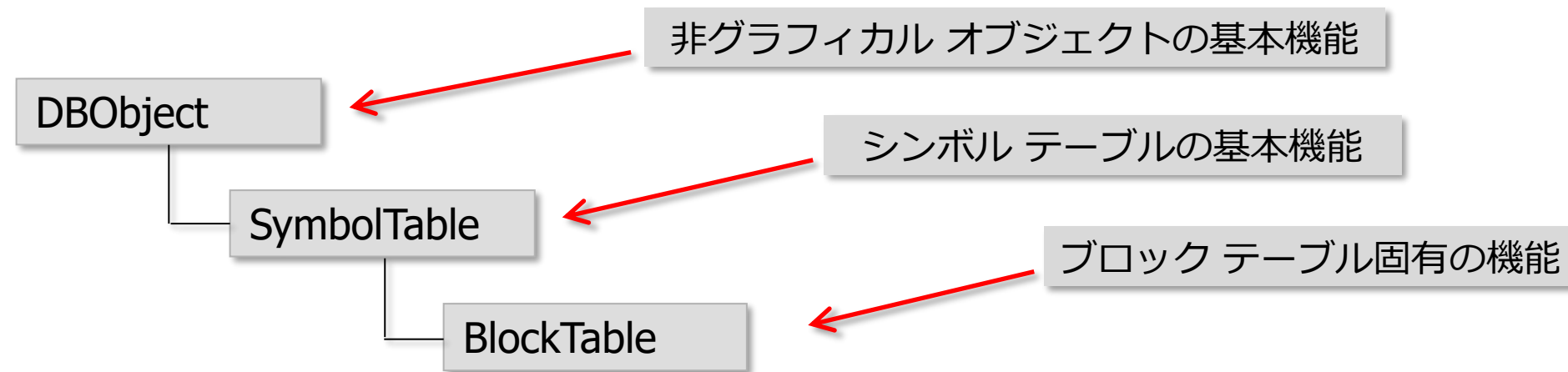
- 代表的なクラス継承の例 1
- Line クラスはグラフィカル クラスの機能を継承
- グラフィカル クラスは非グラフィカル クラス機能を継承



※ 円 (Circle) や 円弧 (Arc) も Curve クラスから 派生

# クラス継承

- 代表的なクラス継承の例 2
- BlockTable クラスはシンボル テーブル機能を継承
- シンボル テーブル クラスは非グラフィカル クラス機能を継承



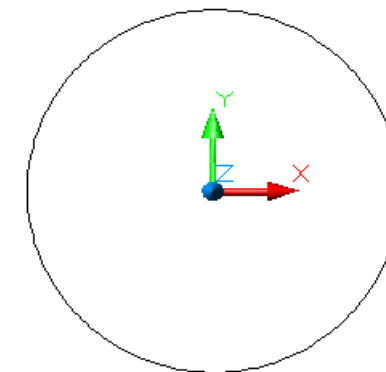
※画層テーブル (LayerTable) や寸法スタイルテーブル (DimStyleTable)、  
字体スタイルテーブル (TextStyleTable) も SymbolTable クラスから 派生

# 階層図についての注意

- 図面データベースの構造
  - メモリ上の図面データベースで**オブジェクトの格納関係を表現**
    - COM（ActiveXオートメーション）のオブジェクト モデルに類似
- クラス ライブラリの階層構造
  - **オブジェクトの継承関係を表現**
  - クラスマップ
    - ObjectARX SDK インストールフォルダ直下：¥classmap¥classmap.dwg

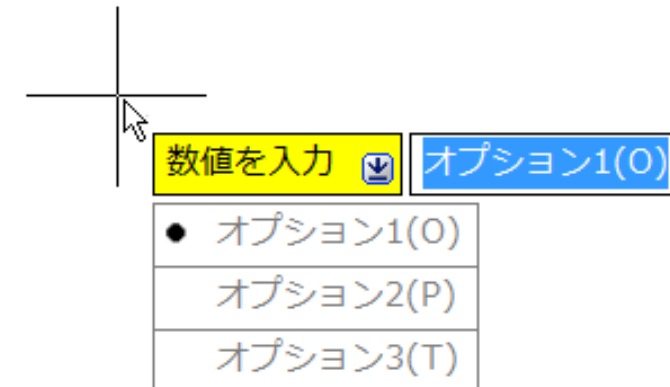
# Lesson 3: ユーザ対話で円を追加

- 円オブジェクト
  - Circle クラスで定義
  - 一般的なユーザ対話で取得すべき情報
    - 中心点 : GetPoint メソッド
    - 半径 : GetDouble メソッド、または、GetDistance メソッド
- 実装コマンド
  - CreateCircle2 コマンド
  - Lesson 2 の内容をコピーして修正を加える



# キーワード入力

- ダイナミック入力に対応する方法
  - Get\*\*\* メソッドでのプロンプト指定方法
  - キーワード群は [ と ] で囲む
  - キーワード群は / で区切る
  - 既定値は < と > で囲む



```
Dim oEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

Dim oIntPmt As PromptIntegerOptions = New PromptIntegerOptions(
    vbCrLf & "数値を入力[オプション1 (0) / オプション2 (P) / オプション3 (T)]", _
    "Option1 option2 option3")
oIntPmt.Keywords.Default = "Option1"

Dim oInt As PromptIntegerResult = oEd.GetInteger(oIntPmt)
If oInt.Status = PromptStatus.Keyword Then
    oEd.WriteMessage(vbCrLf + "キーワードは " + oInt.StringResult.ToString)
ElseIf oInt.Status = PromptStatus.OK Then
    oEd.WriteMessage(vbCrLf + "入力値は " + oInt.Value.ToString)
End If
```

# Lesson 4

## オブジェクトの編集

# Lesson



# VB.NET のエラー処理

- VBA、Visual Basic 6.0 時の一般的なエラー処理
  - On Error Goto *\*label\**
  - On Error Resume Next
- VB.NET
  - 例外処理（Exception Handling）を利用
  - Try ~ Catch 構文
    - マネージ コード全般（VB.NET、C++、C#）共通
    - VB.NET のみ On Error ~ も利用は可能（非推奨）
- Visual Basic 言語でも言語仕様が異なる
  - VBA は Visual Basic 6.0 に準拠
  - .NET 対応した VB.NET は新機軸の対応がされています

# Visual Basic の例外処理

## ■ VB 6.0 の Application オブジェクト取得

```
On Error Resume Next
Dim oApp As AcadApplication
Set oApp = GetObject(, "AutoCAD.Application")      \ AutoCAD インスタンスの取得
If Err Then
    Set oApp = CreateObject("AutoCAD.Application") \ AutoCAD インスタンスの作成
    oApp.Visible = True                          \ AutoCAD ウィンドウを表示状態に変更
End If
```

## ■ VB.NET の Application オブジェクト取得

```
Dim oApp As AutoCAD.AcadApplication
Try
    oApp = GetObject(, "AutoCAD.Application") \ AutoCAD インスタンスの取得
Catch ex As Exception
    oApp = CreateObject("AutoCAD.Application") \ AutoCAD インスタンスの作成
    oApp.Visible = True                      \ AutoCAD ウィンドウを表示状態に変更
End Try
```

# VB.NET のエラー処理

- Try ~ Catch 構文
  - Try ~ Catch 間にエラーが発生すると 例外 が発生！
  - 例外が発生すると Catch の処理が移行
  - Finally はどんな場合でも処理される

```
Dim oDb As Database = HostApplicationServices.WorkingDatabase
Dim oTr As Transaction = oDb.TransactionManager.StartTransaction
Try
    Dim ptStart As Point3d = New Point3d(0.0, 0.0, 0.0)
    Dim ptEnd As Point3d = New Point3d(100.0, 100.0, 0.0)
    Dim oLine As Line = New Line(ptStart, ptEnd)

    Dim oBt As BlockTable = oTr.GetObject(oDb.BlockTableId, OpenMode.ForRead)
    Dim oBtr As BlockTableRecord = oTr.GetObject(oBt.Item("*MODEL_SPACE"), _
                                                OpenMode.ForWrite)

    oBtr.AppendEntity(oLine)
    oTr.AddNewlyCreatedDBObject(oLine, True)
    oTr.Commit()
Catch oEx As Exception
    MsgBox(oEx.ToString())
Finally
    oTr.Dispose()
End Try
```

# トランザクション破棄の簡略化

- Using スコープの利用
  - Using スコープ外でトランザクションが自動的に破棄されます

```
Dim oDb As Database = HostApplicationServices.WorkingDatabase
Using oTr As Transaction = oDb.TransactionManager.StartTransaction
    Try
        Dim ptStart As Point3d = New Point3d(0.0, 0.0, 0.0)
        Dim ptEnd As Point3d = New Point3d(100.0, 100.0, 0.0)
        Dim oLine As Line = New Line(ptStart, ptEnd)

        Dim oBt As BlockTable = oTr.GetObject(oDb.BlockTableId, OpenMode.ForRead)
        Dim oBtr As BlockTableRecord = oTr.GetObject(oBt.Item("*MODEL_SPACE"), _
                                                    OpenMode.ForWrite)

        oBtr.AppendEntity(oLine)
        oTr.AddNewlyCreatedDBObject(oLine, True)
        oTr.Commit()
    Catch oEx As Exception
        MsgBox(oEx.ToString())
    End Try
End Using
```

# 新しいエラー処理 と トランザクションの利点

- クローズし忘れの防止
  - AutoCAD のクラッシュを未然に抑止

```
Try
    oBlockTable = Database.BlockTableId.Open(OpenMode.ForRead)
    oModelSpace = BlockTable(BlockTableRecord.ModelSpace).Open(OpenMode.ForWrite)
    oText = New MText
    oText.SetContents("Hello World !!")
    oModelSpace.AppendEntity(oText)
Finally
    If Not oBlockTable Is Nothing Then
        oBlockTable.Close()
    End If
    If Not oModelSpace Is Nothing Then
        oModelSpace.Close()
    End If
    If Not oText Is Nothing Then
        oText.Close()
    End If
End Try
```

- コミット タイミングの一括処理
  - AutoCAD はクローズ時に始めて図形表示を更新
  - 連続したオープン/クローズ処理による速度遅延を抑止

# オブジェクトの特定

- オブジェクト操作
  - オブジェクトの識別が必要
  - オブジェクト ID ([ObjectId クラス](#))
- 2つの方法
  - AutoCAD上でエンティティを選択する方法
    - 非グラフィカルオブジェクトには適用不可
  - 図面データベースを検索して特定する方法
    - グラフィカル/非グラフィカルオブジェクトに適用可

# 単一エンティティ選択

- 選択メソッド：GetEntity メソッド
  - プロンプト文字による オーバーロード あり
  - プロンプト文字用オブジェクト：PromptEntityOptions
  - 戻り値オブジェクト：PromptEntityResult
  - ObjectId プロパティ にオブジェクト ID を格納
  - 入力判定は Status プロパティ に格納

```
Dim oEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor

Dim oEnt1 As PromptEntityResult = oEd.GetEntity(vbCrLf & "オブジェクトを選択:")
If oEnt1.Status <> PromptStatus.OK Then
    Exit Sub
End If
oEd.WriteMessage(vbCrLf + "オブジェクト ID は " + oEnt1.ObjectId.ToString)
```

# 複数エンティティ選択

- 選択セット取得メソッド：GetSelection メソッド
  - プロンプト文字による オーバーロード あり
  - プロンプト文字用オブジェクト：PromptSelectionOptions
  - 戻り値オブジェクト：PromptSelectionResult
  - Value プロパティ に SelectionSet オブジェクトを格納
  - 入力判定は Status プロパティ に格納

```
Dim oEd As Editor = Application.DocumentManager.MdiActiveDocument.Editor
Dim oEnt1 As PromptSelectionResult = oEd.GetSelection
If oEnt1.Status <> PromptStatus.OK Then
    Exit Sub
End If

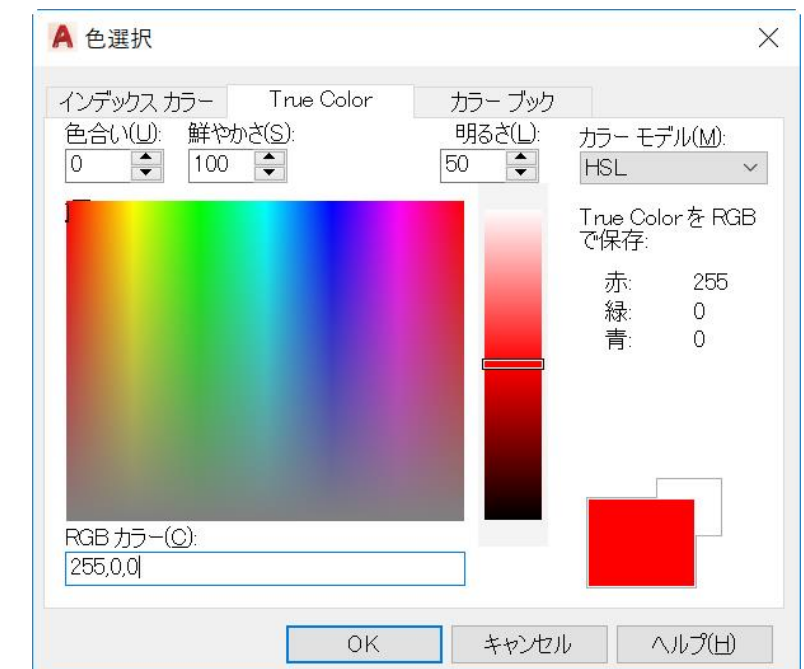
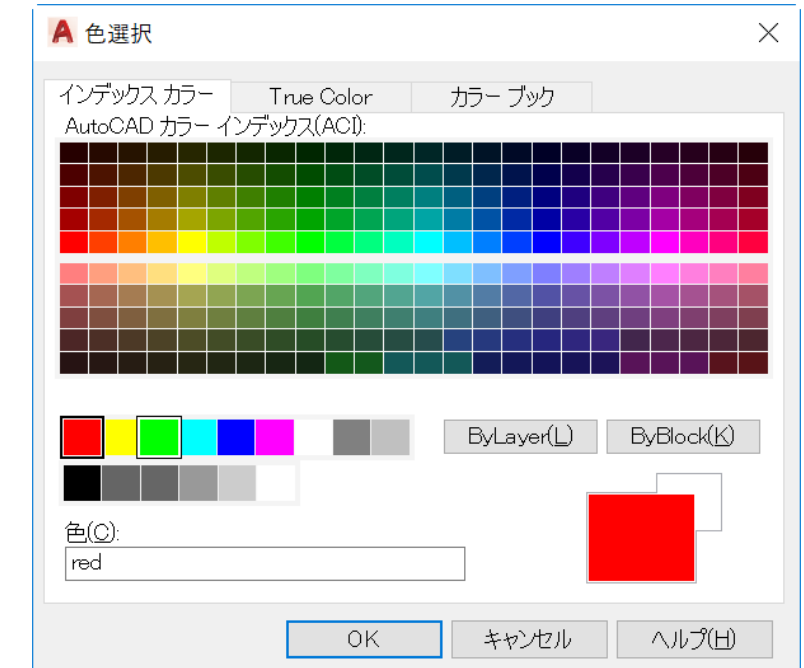
Dim SSet As SelectionSet = oEnt1.Value
Dim objId As ObjectId
For Each objId In SSet.GetObjectIds
    oEd.WriteMessage(vbCrLf + "オブジェクト ID は " + objId.ToString)
Next
```

詳細サンプルは ObjectARX SDK 内の samples¥dotNet¥SelectionSet



# Lesson 4: 選択したオブジェクトの色を変更

- 単一オブジェクトの選択円オブジェクト
  - **GetEntity** メソッド
  - オブジェクト ID を取得
  - トランザクション マネージャでオブジェクトをオープン
  - 色変更アクション: **赤色** に変更
    - 256 色までの色指定の場合
      - **ColorIndex** プロパティ
    - True Colorでの色指定の場合
      - **Color** プロパティ
      - **Color** オブジェクトを使用
- 実装コマンド
  - **ChangeColor** コマンド



# Lesson 5

## パレット ダイアログの作成

# Lesson

# 利用できる代表的なユーザ インタフェース

- .NET Framework クラス ライブラリ から
  - Windows フォーム : [Form クラス](#)
    - ダイアログ ボックス
    - `System.Windows.Forms.Form`
- AutoCAD マネージ クラス ライブラリ から
  - パレット形式のダイアログ ボックス
    - 親パレット : [PaletteSet クラス](#)
    - 子パレット (タブ) : [Palette クラス](#)、または、[UserControl クラス](#)
- その他 AutoCAD 固有のダイアログ
  - 色選択ダイアログ : [ColorDialog クラス](#)

# .NET Framework クラス ライブラリの名前空間

## System.Web

Services

Description

Web サービスを含む Web 関連

Discovery  
Protocols

Caching

Configuration

UI

HtmlControls

WebControls

Security

SessionState

## System.WinForms

Design Windows フォーム関連

## System.Drawing

Drawing

グラフィックス関連

Imaging

Text

## System.Data

データベース (ADO.NET) 関連

Design

SQLTypes

## System.Xml

XSLT

XML 処理関連

XPath

## System

Collections

Configuration

Diagnostics

Globalization

IO

Nls

Reflection

Resources

Security

WebControls

Text

Threading

Runtime

Services

Remoting

Serialization

データ型、属性、数値演算など基本機能関連

# AutoCAD マネージ クラス ライブラリの名前空間

**Autodesk.AutoCAD.GraphicsInterface**

ベクトル描画機構関連

WorldDraw

ViewportDraw

SubEntity Traits

**Autodesk.AutoCAD.PlottingServices**

印刷サービス関連

PlotConfig

PlotEngine

PlotConfig

PlotEngine

**Autodesk.AutoCAD.Windows**

ユーザインタフェース関連

CommandFlags

DynamicLinker

Exception

RuntimeSystem

**Autodesk.AutoCAD.EditorInput**

選択、入力等ユーザ対話関連

Editor

GetSelection

GetPoint

GetEntity

Keyword

Jig

**Autodesk.AutoCAD.Runtime**

例外処理など基本機能関連

CommandFlags

DynamicLinker

Exception

RuntimeSystem

**Autodesk.AutoCAD.Geometry**

幾何演算関連

Point2d

Point3d

Vector2d

Vector3d

Matrix2d

Matrix3d

**Autodesk.AutoCAD.DatabaseServices**

グラフィカル、非グラフィカルオブジェクト等図面データベース関連

Line

Arc

Circle

BlockReference

AttributeDefinition

Database

ObjectId

BlockTable

DimStyleTable

TextStyleTable

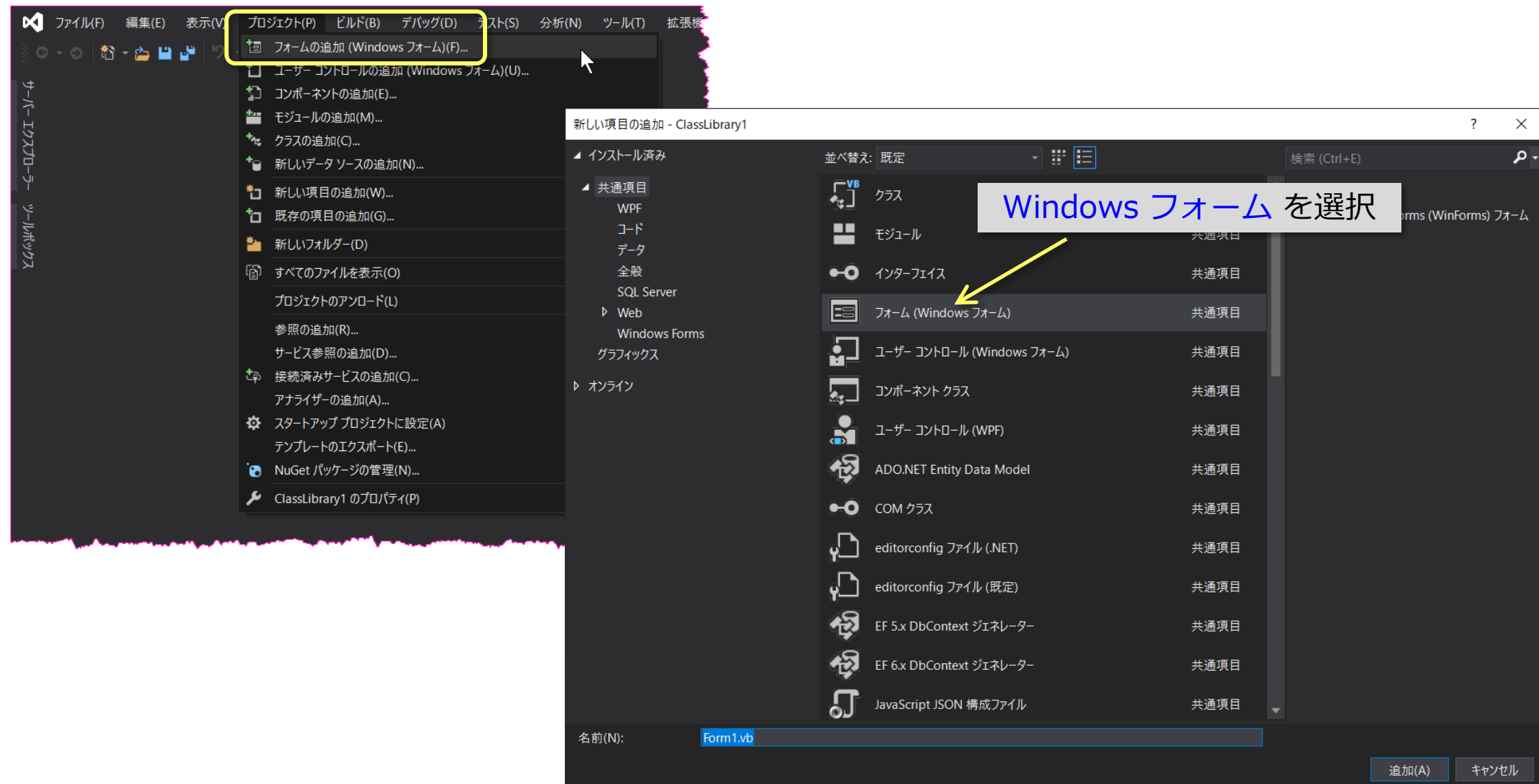
BlockTableRecord

DimStyleTableRecord

TextStyleTableRecord

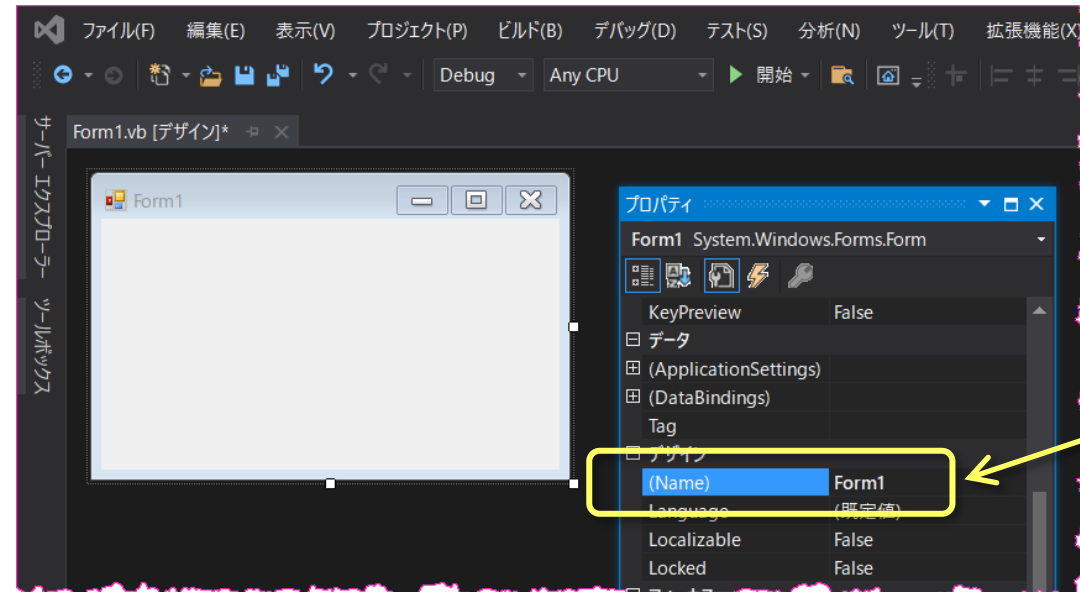
# Windows フォームの追加手順

## 1. プロジェクトに Windows フォームを追加



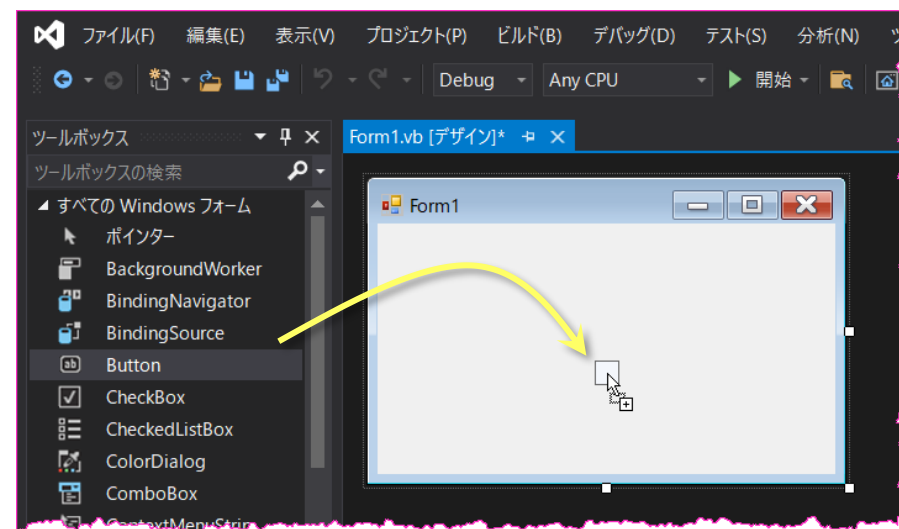
# Windows フォームの追加手順

## 2. 追加されたフォームの Name を確認 (Form1 と仮定)

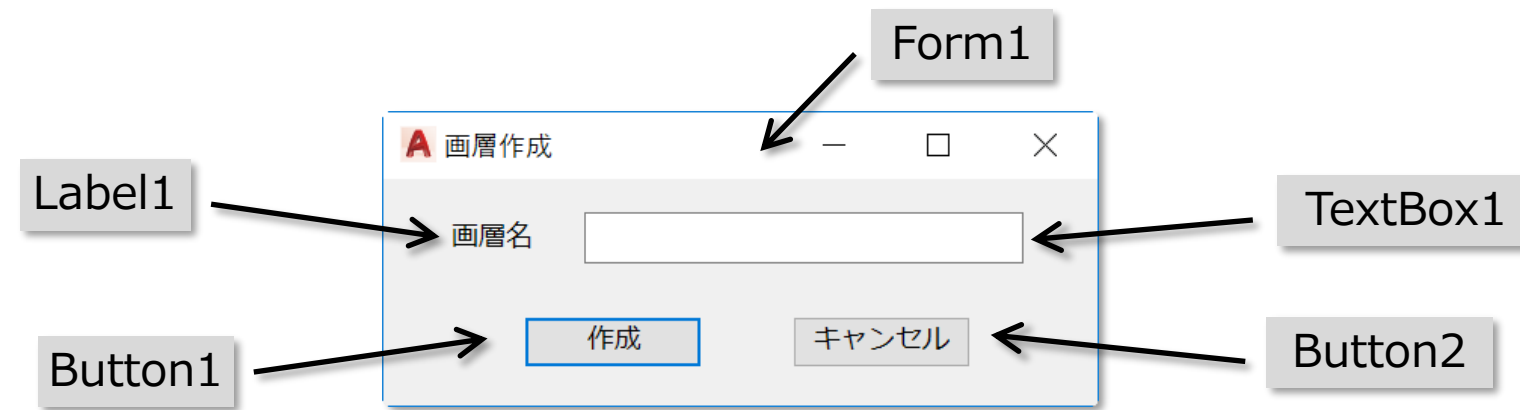


プログラムで使用するクラス名  
⇒ 新しいクラスを派生 !!

## 3. 必要なコントロールをドラッグ&ドロップでフォーム上に配置



# Windows フォームの追加手順



## 4. フォーム (Form1.vb) のコード ウィンドウに Imports を追記

- ・ コマンド定義モジュール Class1.vb とは別モジュールのため

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.ApplicationServices
```

```
Public Class Form1
    :
```

## 5. キャンセル ボタン (Button2) のプロシージャを記述

```
Private Sub Button2_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) Handles Button2.Click

    Me.Hide()
End Sub
```



# Windows フォームの追加手順

## ■ 画層作成 ボタン（Button1）のプロシージャを記述

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _  
    System.EventArgs) Handles Button1.Click  
  
    Dim oDb As Database = HostApplicationServices.WorkingDatabase  
    Dim oTr As Transaction = oDb.TransactionManager.StartTransaction  
    Try  
        Dim oLt As LayerTable = oTr.GetObject(oDb.LayerTableId, OpenMode.ForWrite)  
        Dim oLtr As LayerTableRecord  
        If Not oLt.Has(TextBox1.Text) Then  
            oLtr = New LayerTableRecord  
            oLtr.Name = TextBox1.Text  
            oLt.Add(oLtr)  
            oTr.AddNewlyCreatedDBObject(oLtr, True)  
        Else  
            MsgBox("既に画層が存在しています")  
        End If  
        oTr.Commit()  
    Catch oEx As Exception  
        MsgBox(oEx.ToString())  
    Finally  
        oTr.Dispose()  
    End Try  
    Me.Hide()  
  
End Sub
```

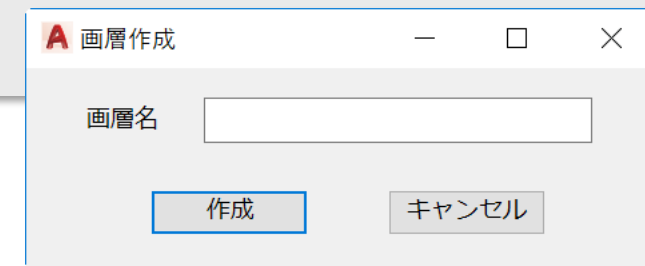
# Windows フォームの追加手順

## ■ ダイアログ表示用のコマンドを定義 (Class1.vb と仮定)

```
<CommandMethod("ShowDialog")> _  
Public Shared Sub ShowDialog()  
  
    Dim oForm As Form1 = New Form1  
    Application.ShowModalDialog(oForm)  
    oForm.Dispose()  
  
End Sub
```

処理はフォーム側で実装されているので、コマンド側でダイアログ終了時のボタン判定をしない

通常、ダイアログは Hide メソッドで非表示になっているだけなので、ここで明示的にメモリ上から破棄



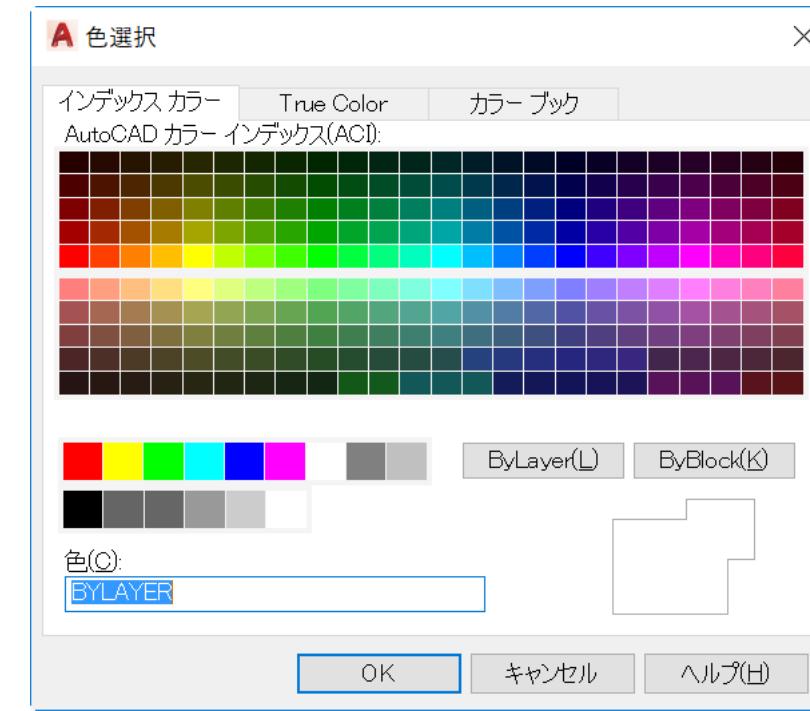
⚠ AutoCAD 内では System.Windows.Forms.Form を使った フォームでも Form.ShowDialog メソッドは利用できません

 ShowModalDialog メソッド を代わりに使います

- Autodesk.AutoCAD.ApplicationServices.Application.ShowModalDialog

# コマンド定義側でのダイアログ終了判定

- 色選択ダイアログの場合
  - ColorDialog クラス



```
<CommandMethod("ShowDialog")> _  
Public Shared Sub ShowDialog()
```

```
    Dim oForm As ColorDialog = New ColorDialog  
    Dim oResult As System.Windows.Forms.DialogResult = oForm.ShowDialog()  
    If (oResult = Windows.Forms.DialogResult.OK) Then  
        MsgBox(oForm.Color.ToString)  
    End If
```

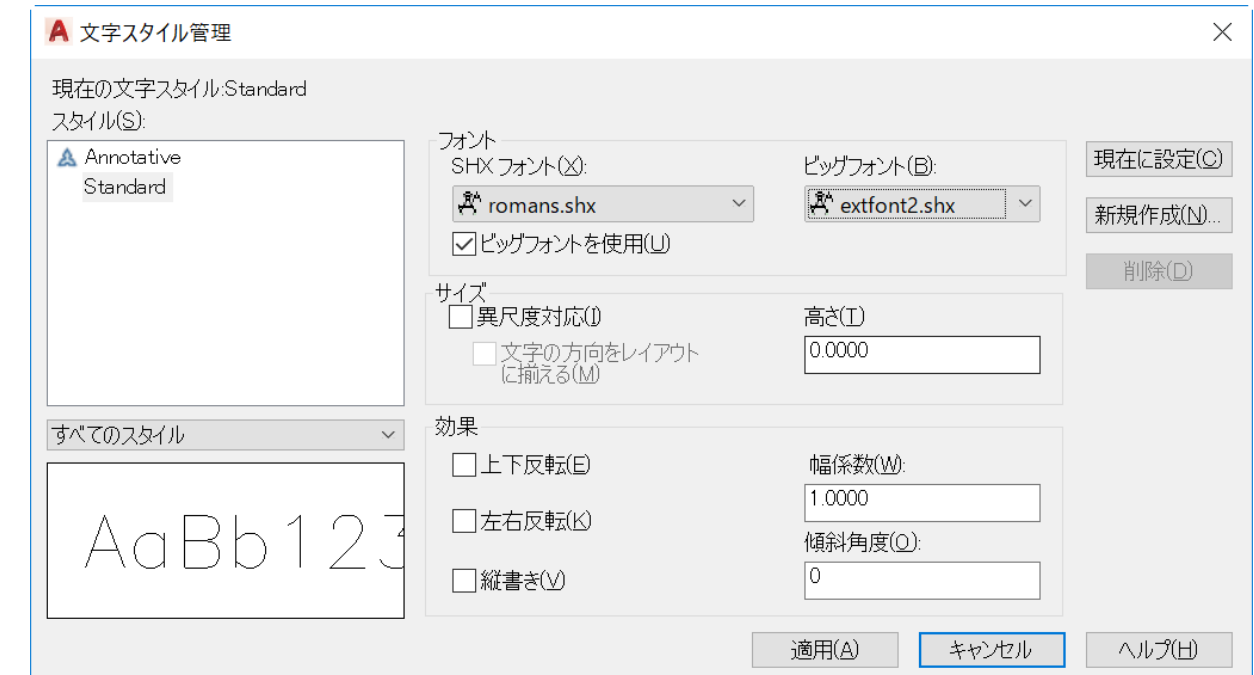
```
End Sub
```

ボタン毎に設定された DialogResult 値が返される

ColorDialog は AutoCAD が管理するので  
Dispose でメモリ上から破棄する必要がない

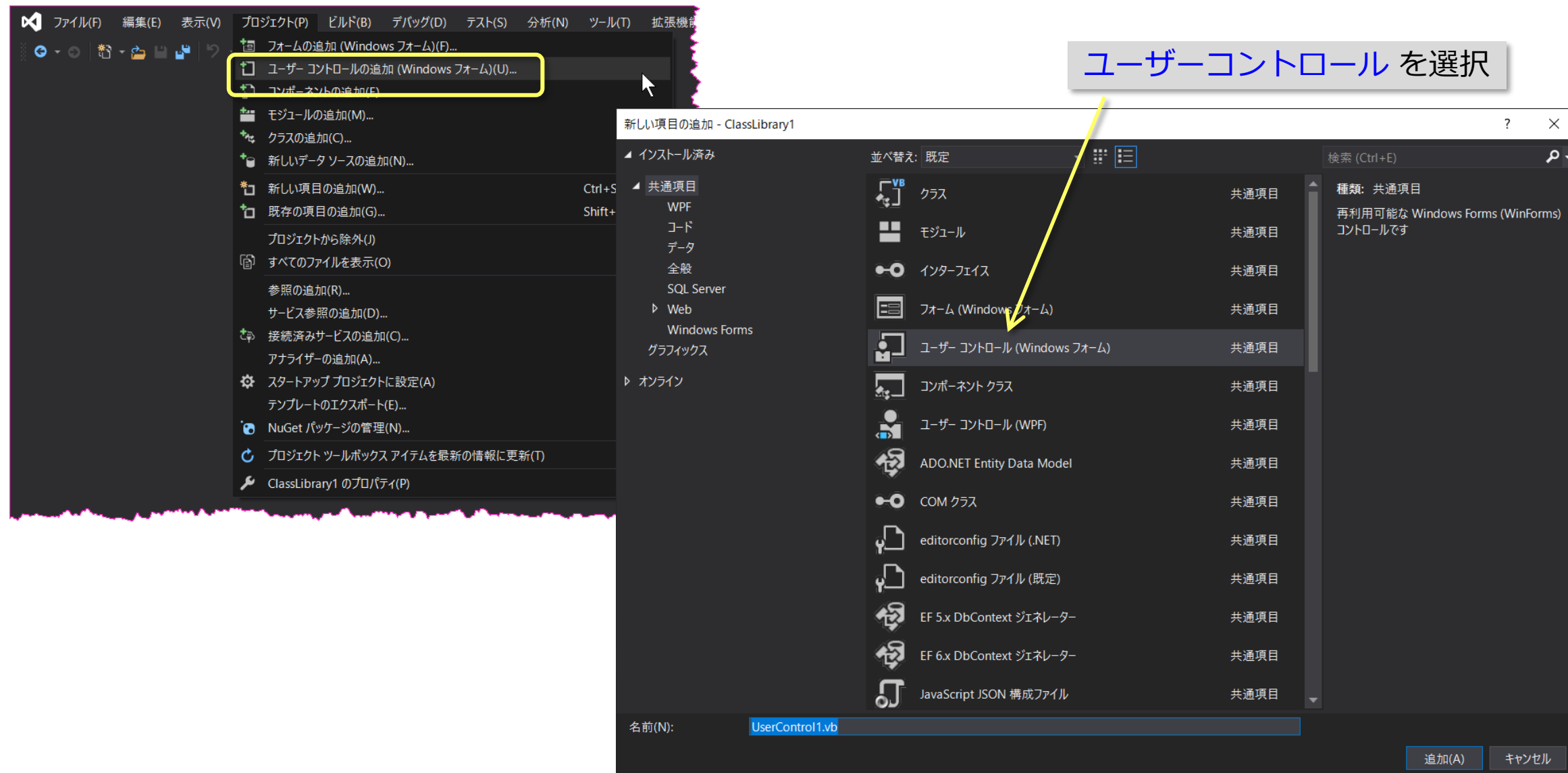
# Lesson 5: パレット ダイアログの作成

- パレットを使った文字スタイルの作成
  - [TextStyleTableRecord](#) クラス
  - スタイルが既に存在しているかチェック！
    - [SymbolTable.Has](#) メソッド
  - 指定するフォント
    - SHX フォント : romans.shx : [TextStyleTableRecord.FileName](#) プロパティ
    - ビックフォント : extfont2.shx : [TextStyleTableRecord.BigFontFileName](#) プロパティ
- 入力パレットの作成
  - [PaletteSet](#) クラス と作成した [UserControl](#) クラス
  - 配置するコントロール：
    - 文字スタイル名 - [Label](#)、文字スタイル名入力 - [TextBox](#)、作成とキャンセル - [Button](#)
- 実装コマンド
  - [ShowPalette](#) コマンド



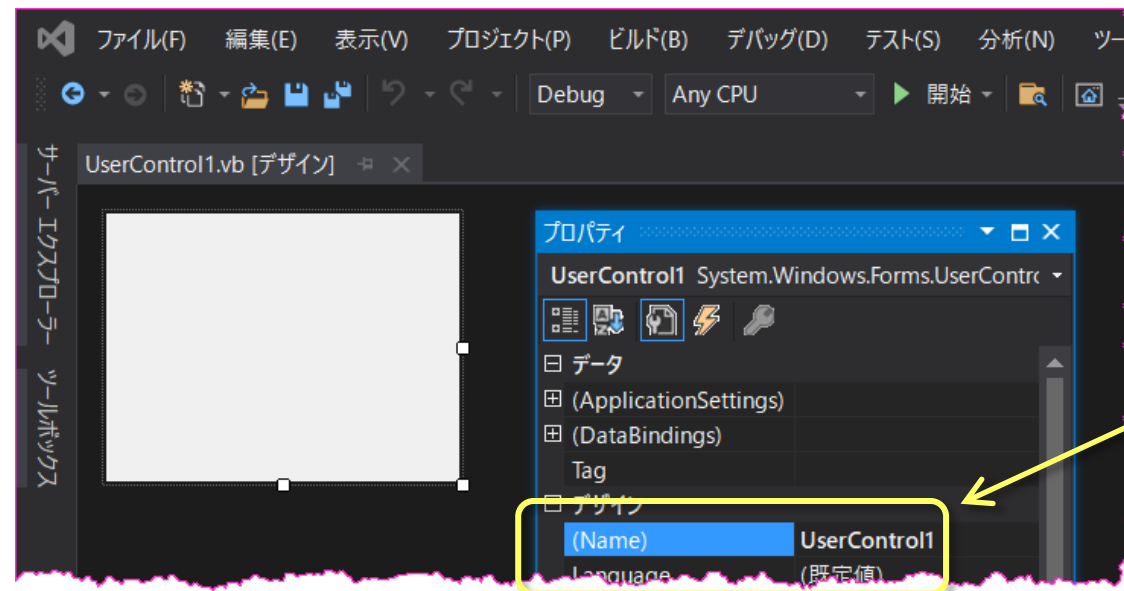
# Lesson 5: パレット ダイアログの作成

## 1. プロジェクトに ユーザーコントロール を追加



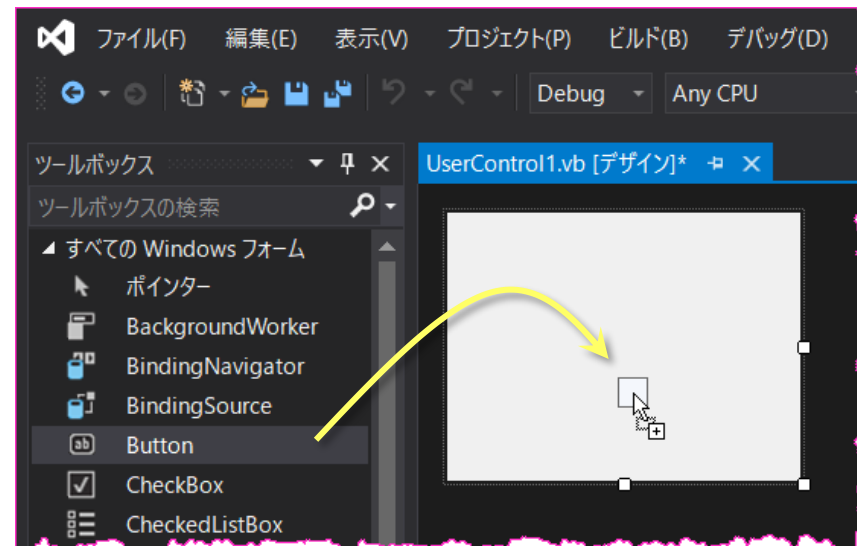
# Lesson 5: パレット ダイアログの作成

## 2. 追加されたユーザーコントロールの Name を確認 (UserControl1 と仮定)

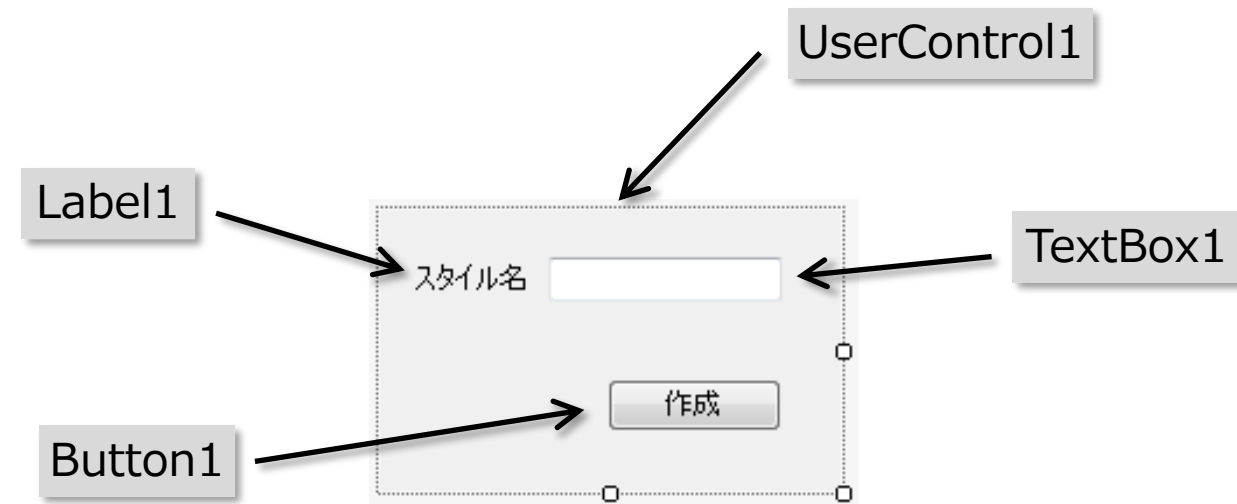


プログラムで使用するクラス名  
⇒ 新しいクラスを派生 !!

## 3. 必要なコントロールをドラッグ&ドロップでコントロール上に配置



# Lesson 5: パレット ダイアログの作成



- ## 4. ユーザーコントロールのコード ウィンドウに Imports を追記
- コマンド定義モジュール Class1.vb とは別モジュールのため

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.Windows

Public Class UserControl1
    :
```

# Lesson 5: パレット ダイアログの作成

## ■ 文字スタイル作成 ボタン (Button1) のプロシージャを記述

```
Dim oDb As Database = HostApplicationServices.WorkingDatabase
Dim oTr As Transaction = oDb.TransactionManager.StartTransaction
Try
    Dim oDocLock As DocumentLock = _
        Application.DocumentManager.MdiActiveDocument.LockDocument
    Dim oTt As TextStyleTable = oTr.GetObject(oDb.TextStyleTableId, _
        OpenMode.ForWrite)

    Dim oTtr As TextStyleTableRecord
    If Not oTt.Has(textBox1.Text) Then
        oTtr = New TextStyleTableRecord
        oTtr.Name = textBox1.Text
        oTtr.FileName = "romans.shx"
        oTtr.BigFontFileName = "extfont2.shx"
        oTt.Add(oTtr)
        oTr.AddNewlyCreatedDBObject(oTtr, True)
    Else
        MsgBox("既に文字スタイルが存在しています")
    End If
    oTr.Commit()
    oDocLock.Dispose()
Catch oEx As Exception
    MsgBox(oEx.ToString())
Finally
    oTr.Dispose()
End Try
```



# Lesson 5: パレット ダイアログの作成

## 6. パレットセットとパレット用変数の宣言 (Class1.vb と仮定)

- パレット表示コマンドのあるモジュール内

```
Imports Autodesk.AutoCAD.Runtime
Imports Autodesk.AutoCAD.DatabaseServices
Imports Autodesk.AutoCAD.ApplicationServices
Imports Autodesk.AutoCAD.EditorInput
Imports Autodesk.AutoCAD.Geometry
Imports Autodesk.AutoCAD.Colors
Imports Autodesk.AutoCAD.Windows

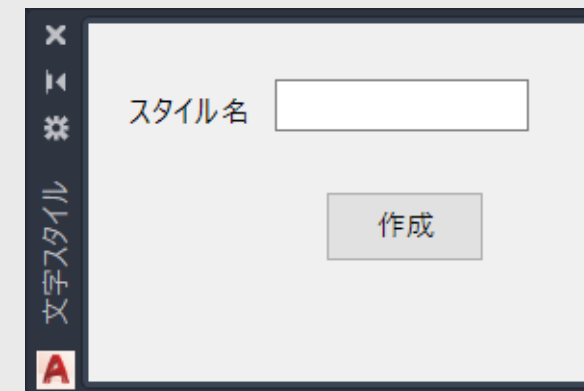
Public Class Class1

    ' パレットセットとパレット用変数の宣言
    Public Shared oP As UserControl1 = Nothing
    Public Shared oPs As Autodesk.AutoCAD.Windows.PaletteSet = Nothing
```

# Lesson 5: パレット ダイアログの作成

## 7. パレット表示用のコマンドを定義 (Class1.vb と仮定)

```
<CommandMethod("ShowPalette")> _  
Public Shared Sub MyPalette()  
    ' パレットセットの作成  
    If oPs Is Nothing Then  
        oPs = New Autodesk.AutoCAD.Windows.PaletteSet("文字スタイル")  
        oPs.Style = PaletteSetStyles.ShowPropertiesMenu Or _  
            PaletteSetStyles.ShowAutoHideButton Or _  
            PaletteSetStyles.ShowCloseButton Or _  
            PaletteSetStyles.Snappable  
        oPs.MinimumSize = New System.Drawing.Size(300, 200)  
        oPs.Visible = True  
    Else  
        oPs.Visible = True  
    End If  
    ' パレットの作成  
    If oPs.Count = 0 Then  
        oP = New UserControl1  
        oPs.Add("文字スタイル作成", oP)  
    End If  
    oPs.Dock = DockSides.None  
    oPs.Size = New System.Drawing.Size(300, 200)  
  
End Sub
```



# ダイアログ ボックス (UI) の種類と注意

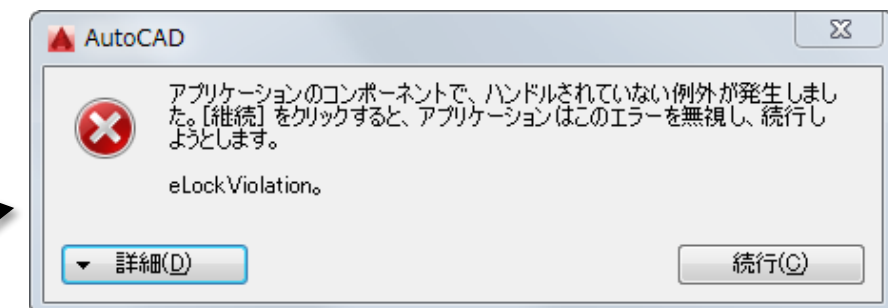
## ■ モーダル ダイアログ

- ダイアログを閉じない親ウィンドウを操作できない
- ダイアログ表示中に図面ウィンドウの切り替えは不可
- ダイアログ側で実装したプログラムの反映タイミング  
→ ダイアログを閉じたとき

## ■ モードレス ダイアログ

- ダイアログを閉じなくても親ウィンドウを操作できる
- ダイアログ表示中に図面ウィンドウの切り替えが可能
- ダイアログ側で実装したプログラムの反映タイミング → 反映対象の図面は?
  - 対象不定のアクションがランタイムエラーを誘発

eLockViolation



# モードレス UI からの図面データベース アクセス

```
Dim oDb As Database = HostApplicationServices.WorkingDatabase
Dim oTr As Transaction = oDb.TransactionManager.StartTransaction
Try
    Dim oDocLock As DocumentLock
    oDocLock = Application.DocumentManager.MdiActiveDocument.LockDocument
    Dim oTt As TextStyleTable = oTr.GetObject(oDb.TextStyleTableId, _
                                                OpenMode.ForWrite)

    Dim oTtr As TextStyleTableRecord
    If Not oTt.Has(TextBox1.Text) Then
        oTtr = New TextStyleTableRecord
        oTtr.Name = TextBox1.Text
```

ドキュメントをロック !!

ドキュメントの ロック が対象図面を特定します  
(ドキュメント=図面ウィンドウは Document クラスで表現)

```
Else
    MsgBox ("既に字体スタイルが存在しています")
End If
oTr.Commit()
oDocLock.Dispose()

Catch oEx As Exception
    MsgBox(oEx.ToString())
Finally
    oTr.Dispose()
End Try
```

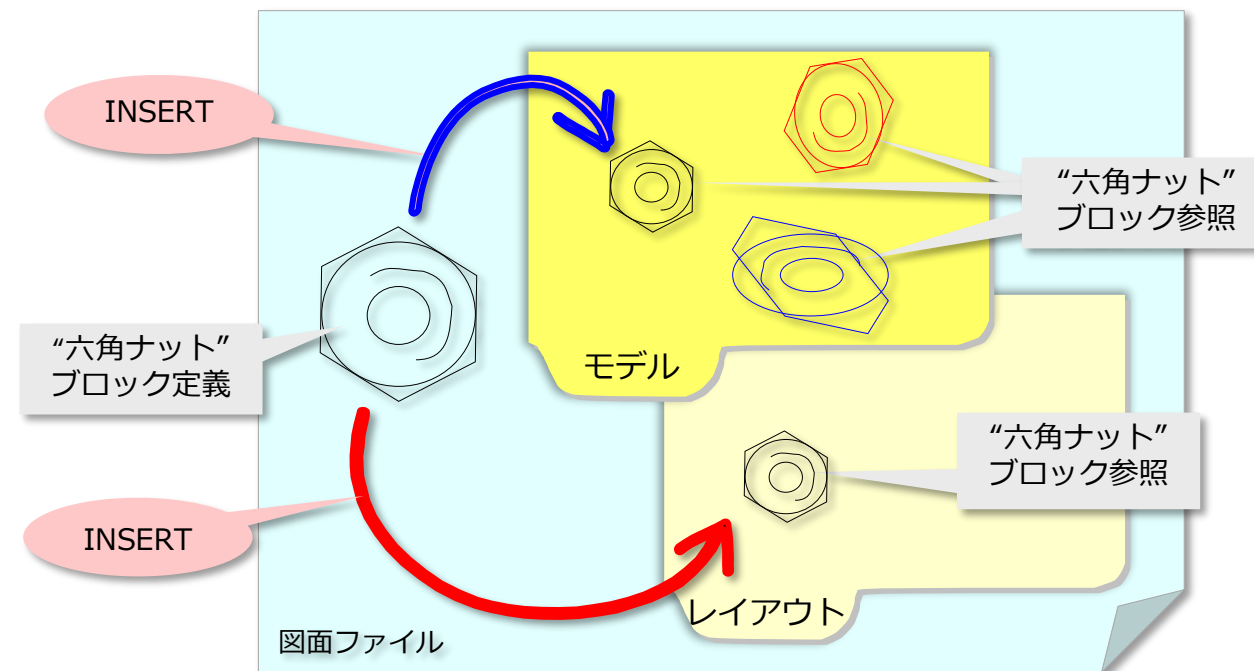
# Lesson 6

## 非グラフィカル オブジェクトとの連携

# Lesson

# ブロック定義とブロック参照の関係

- **ブロック定義**：定義情報なので図面上には表示されない
  - BlockTableRecord クラス
  - モデル空間やペーパー空間に配置されるブロックの定義
- **ブロック参照**：個々のブロック図形として図面上に表示される
  - BlockReference クラス
  - 実際に配置されるブロック定義の複製、または、インスタンス



# ブロック定義とブロック参照の関係

- ブロック参照がブロック定義を参照する場合
  - コマンドや VBA の場合
    - ブロック名（文字列）で指定
- .NET API の場合
  - オブジェクト ID で指定（ObjectID クラス）
    - AutoCAD オブジェクトはオブジェクト ID を持つ
    - オブジェクト の ObjectID プロパティから取得

# ブロック定義とブロック参照の作成

```
Dim oBt As BlockTable = oTr.GetObject(oDb.BlockTableId, OpenMode.ForWrite)
Dim oBlkDef As BlockTableRecord
If Not oBt.Has("Test") Then

    Dim oCirc As Circle = New Circle(Point3d.Origin, Vector3d.ZAxis, 10.0)
    oBlkDef = New BlockTableRecord
    oBlkDef.Name = "Test"
    oBt.Add(oBlkDef) ' ブロック定義を先に DB 登録 (eNotInDatabase 防止)
    oTr.AddNewlyCreatedDBObject(oBlkDef, True)
    oBlkDef.AppendEntity(oCirc)
    oTr.AddNewlyCreatedDBObject(oCirc, True)

Else
    oBlkDef = oTr.GetObject(oBt.Item("Test"), OpenMode.ForWrite)
End If

Dim oPt As Point3d
oPt = New Point3d(100.0, 100.0, 0.0)
Dim oBlkRef As BlockReference = New BlockReference(oPt, oBlkDef.ObjectId)

Dim oCurBtr As BlockTableRecord = oTr.GetObject(oBt.Item("*MODEL_SPACE"), _
                                                OpenMode.ForWrite)

oCurBtr.AppendEntity(oBlkRef)
oTr.AddNewlyCreatedDBObject(oBlkRef, True)
oTr.Commit()
```

: <Try スコープ抜粋>

ブロック定義

ブロック参照

ブロック定義の オブジェクトID で参照を指定



# グラフィカル オブジェクトと非グラフィカル オブジェクトの関係

- ブロック参照 と ブロック定義
  - [BlockReference クラス](#) と [BlockTableRecord クラス](#)
  - BlockReference.BlockTableRecord プロパティに BlockTableRecord.ObjectId 値を設定
- マルチテキスト文字 と 文字スタイル
  - [MText クラス](#) と [TextStyleTableRecord クラス](#)
  - MText.TextStyle プロパティに TextStyleTableRecord.ObjectId 値を設定
- 寸法 と 寸法スタイル
  - [Dimension 以下派生クラス](#) と [DimStyleTableRecord クラス](#)
  - Dimension.DimensionStyle プロパティに DimStyleTableRecord.ObjectId 値を設定

# グラフィカル オブジェクト と 非グラフィカル オブジェクト の関係

- 図形 と 画層

- Entity 以下の派生 クラス と LayerTableRecord クラス
- Entity.LayerId プロパティに LayerTableRecord.ObjectId 値を設定

- 図形 と 線種

- Entity 以下の派生 クラス と LinetypeTableRecord クラス
- Entity.LinetypeId プロパティに LinetypeTableRecord.ObjectId 値を設定

# Lesson 6: 作成した円の画層を指定

- Lesson 3 の内容をコピーして画層処理を追加
  - 新しい画層 “トレーニング”
  - `LayerTableRecord` を `LayerTable` に追加
  - “トレーニング” 画層の `ByLayer` 色を 黄色 に設定
- 円を “トレーニング” 画層に作画
  - 円の `LayerId` プロパティ に “トレーニング” 画層の `ObjectId` 値を設定
- 実装コマンド
  - `AttachLayer` コマンド

# Lesson 7

## イベントハンドリング

# Lesson

# AutoCAD イベントを監視

- .NET Framework のデリゲートを利用
- AutoCAD上で発生するイベントの例
  - コマンドの実行
  - ドキュメントのクローズ
  - 図面データベースへのオブジェクトの追加 …など
- AutoCAD上で発生するイベントの監視
  - コマンドの監視：エディタ ハンドラ ← エディタ機能はドキュメント毎であるため
  - ドキュメントの監視：ドキュメント ハンドラ、  
ドキュメントコレクション ハンドラ
  - 図面データベースの監視：データベース ハンドラ

# ドキュメント イベント

## ■ Document オブジェクトで処理可能な主なイベント

BeginDocumentClose イベント	ドキュメントが閉じられようとしている
CloseAborted イベント	ドキュメントを閉じる処理が中断された
CloseWillStart イベント	ドキュメントを閉じる処理が開始されようとしている
CommandCancelled イベント	コマンドがキャンセルされた
CommandEnded イベント	コマンドが終了した
CommandFailed イベント	コマンドが失敗した
CommandWillStart イベント	コマンドが開始されようとしている
LispCancelled イベント	LISP がキャンセルされた
LispEnded イベント	LISP が終了した
LispWillStart イベント	LISP が開始されようとしている
UnknownCommand イベント	不明なコマンドが入力された

# ドキュメント コレクション イベント

## ■ DocumentCollection オブジェクトで処理可能な主なイベント

DocumentActivated イベント	ドキュメントがアクティブになった
DocumentBecameCurrent イベント	ドキュメントがカレント ドキュメントになろうとしている
DocumentCreated イベント	ドキュメントが作成された
DocumentCreateStarted イベント	ドキュメントの作成が開始された
DocumentDestroyed イベント	ドキュメントが破棄された
DocumentToBeActivated イベント	ドキュメントがアクティブになろうとしている
DocumentToBeDeactivated イベント	ドキュメントが非アクティブになろうとしている
DocumentToBeDestroyed イベント	ドキュメントが破棄されようとしている

# データベース イベント

## ■ Database オブジェクトで処理可能な主なイベント

BeginInsert イベント	ブロック挿入が開始されようとしている
BeginSave イベント	図面保存が開始されようとしている
DatabaseConstructed イベント	データベースが構築された
DatabaseToBeDestroyed イベント	データベースが破棄されようとしている
ObjectAppended イベント	オブジェクトが追加された
ObjectErased イベント	オブジェクトが削除された
ObjectModified イベント	オブジェクトが変更された
ObjectReappended イベント	オブジェクトが再追加された(アンドゥ後のリドゥ)
ObjectUnappended イベント	オブジェクトの追加が取り消された(アンドゥ)



# イベント ハンドラの登録と解除

- イベント ハンドラの登録
  - [AddHandler ステートメント](#)
  - AddHandler オブジェクト, イベント, AddressOf イベント ハンドラ
- イベント ハンドラの解除
  - [RemoveHandler ステートメント](#)
  - RemoveHandler オブジェクト, イベント, AddressOf イベント ハンドラ
- 各ハンドラ
  - イベント ハンドラの引数として関連要素が通知される
  - イベント ハンドラは戻り値を返す機能はありません

# イベント ハンドラの例

・ データベース イベント ハンドラの作成

```
<CommandMethod("AddDBEvent")> _  
Public Sub AddDBEvent()  
    Dim oDb As Database = HostApplicationServices.WorkingDatabase  
    AddHandler oDb.ObjectAppended, AddressOf ObjectAppendedHandler  
    AddHandler oDb.ObjectErased, AddressOf ObjectErasedHandler  
End Sub
```

・ データベース イベント ハンドラの削除

```
<CommandMethod("RemDBEvent")> _  
Public Sub RemDBEvent()  
    Dim oDb As Database = HostApplicationServices.WorkingDatabase  
    RemoveHandler oDb.ObjectAppended, AddressOf ObjectAppendedHandler  
    RemoveHandler oDb.ObjectErased, AddressOf ObjectErasedHandler  
End Sub
```

・ オブジェクト追加後の呼び出しハンドラ

```
Private Shared Sub ObjectAppendedHandler(ByVal sender As Object, _  
                                         ByVal e As ObjectEventArgs)  
    MsgBox(e.DBObject.GetRXClass.DxfName & " オブジェクト追加後の呼び出しハンドラ")  
End Sub
```

・ オブジェクト削除後の呼び出しハンドラ

```
Private Shared Sub ObjectErasedHandler(ByVal sender As Object, _  
                                       ByVal e As ObjectErasedEventArgs)  
    MsgBox(e.DBObject.GetRXClass.DxfName & " オブジェクト削除後の呼び出しハンドラ")  
End Sub
```

# Lesson 7: イベント ハンドラ

- データベース イベントの監視
  - オブジェクトの追加と削除を監視
  - オブジェクト タイプをコマンド プロンプトに表示
- 実装コマンド
  - データベース イベントの監視開始 : [AddDBEvent コマンド](#)
  - データベース イベントの監視終了 : [RemDBEvent コマンド](#)

図面データベースとコマンドの関係に注意 !!  
その他、懸念点も検証してください

# Lesson 8

## カスタム データ

# Lesson

# 図面ファイル内での独自データの付加方法の考察

## ■ ブロック属性

- 👍 ユーザ操作で属性を付加することができる
- 👎 ユーザ操作で属性を消されてしまう可能性がある
- 👎 付加できるのはブロック単位のみでバイナリデータの扱いは不可能

## ■ カスタム オブジェクト

- 👍 あらゆるデータを内包できる自由度の高いオブジェクト作成が可能
- 👍 ユーザ操作でデータが削除されてしまう恐れはない
- 👍 オブジェクトの振る舞い（動作、挙動）も同時に定義することができる
- 👍 グラフィカル オブジェクトと非グラフィカル オブジェクトの両者を実装可能
- 👎 ObjectARX でしか定義できない（.NET API のみの実現は不可能）
- 👎 定義アドオンがロードされない環境ではプロキシ オブジェクトになってしまう

# 図面ファイル内での独自データの付加方法の考察（つづき）

## ■ 拡張エンティティ データ

- 👍 バイナリを含むあらゆる情報をあらゆるオブジェクトに付加できる
- 👍 ユーザ操作でデータが削除されてしまう恐れはない
- 👎 1 オブジェクトへの付加サイズ総量が 16 KByte を越えられない

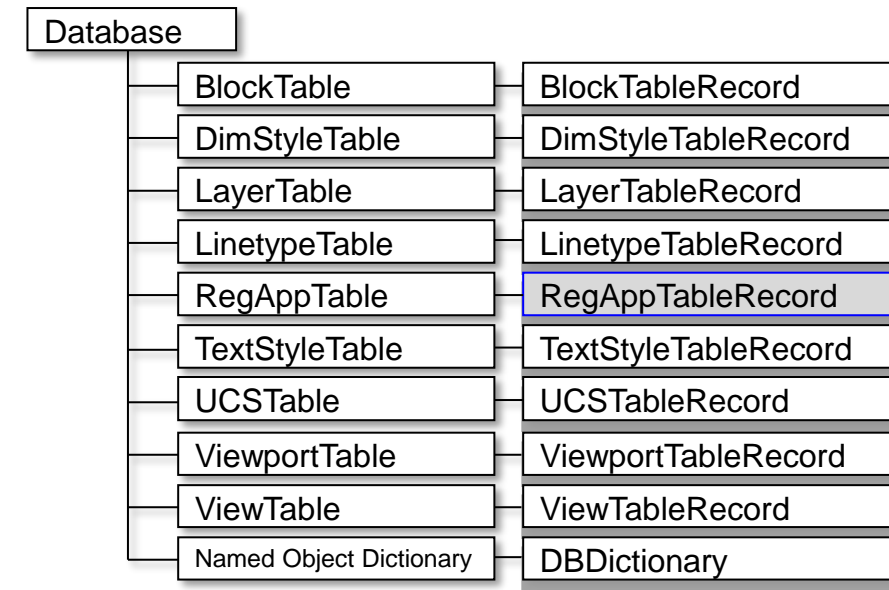
## ■ 拡張ディクショナリ（拡張レコード）

- 👍 バイナリを含むあらゆる情報をあらゆるオブジェクトに付加できる
- 👍 ユーザ操作でデータが削除されてしまう恐れはない
- 👍 1 オブジェクトへの付加サイズ総量制限は特にない（オブジェクト全体で4GByte）
- 👎 バイナリ データは 127 Byte 単位に分割しなければならない
- 👎 文字列領域 1 つに保存可能な最大文字数は 65535 文字

# 拡張エンティティ データとは？

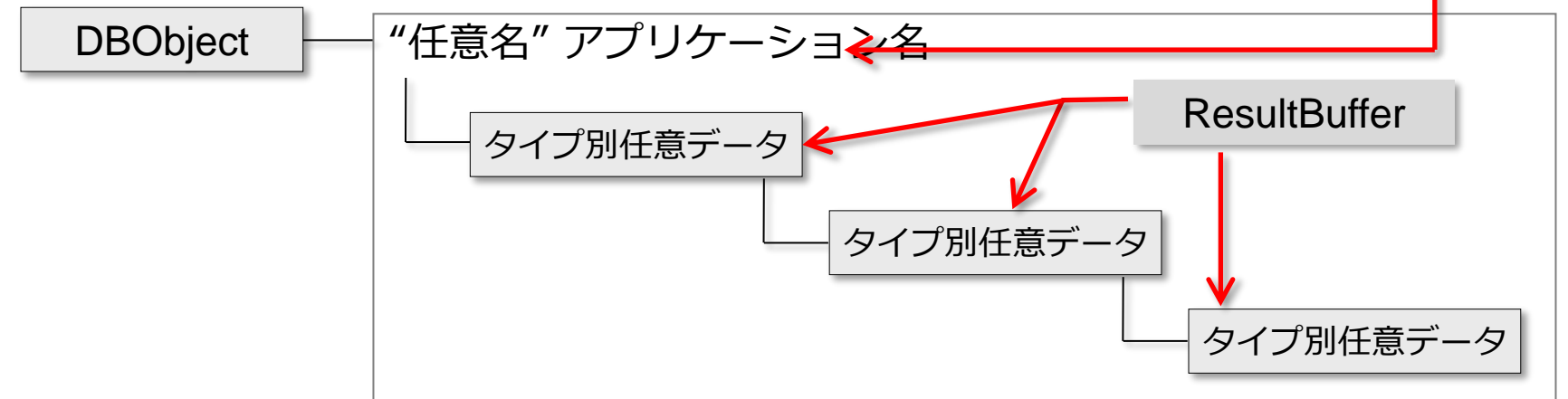
- 拡張エンティティ データのアプリケーション名は事前登録制

- 任意名 を付加してデータ識別
- RegAppTableRecord



- 実際に付加するデータは DXF コード毎に用意したデータ チェイン

- ResultBuffer



# 拡張エンティティ データとは？

- 任意のオブジェクトへ任意の順序で任意の情報を付加
  - 拡張エンティティ データの制御はアプリケーションまかせ
  - AutoCAD はメンテナンスしない
- 付加データは種類別に DXF コードが決まっている
  - **-3** : 拡張エンティティ データ識別子
  - **1000** : 文字データ
  - **1001** : アプリケーション名
  - **1002** : コントロール文字 (“{” または “}”)
  - **1003** : 画層名
  - **1004** : バイナリデータ
  - **1005** : ハンドル番号
  - **1010** : 3次元点
  - **1011** : ワールド空間の位置
  - **1012** : ワールド空間の変位
  - **1013** : ワールド方向
  - **1040** : 実数
  - **1041** : 距離実数
  - **1042** : 尺度係数
  - **1070** : 整数
  - **1071** : 長整数



# 拡張エンティティ データの付加

- アプリケーション名は付加オブジェクト インスタンスで共用可能

```
Dim oEnt As Entity
oEnt = oTr.GetObject(oRst.ObjectId, OpenMode.ForWrite)

Dim oTbl As RegAppTable
oTbl = oTr.GetObject(oDb.RegAppTableId, OpenMode.ForWrite, False)
If Not oTbl.Has("ST_Bolt") Then
    Dim oRec As RegAppTableRecord
    oRec = New RegAppTableRecord
    oRec.Name = "ST_Bolt"
    oTbl.Add(oRec)
    oTr.AddNewlyCreatedDBObject(oRec, True)
End If

oEnt.XData = New ResultBuffer(New TypedValue(1001, "ST_Bolt"), _
    New TypedValue(1002, "{"), _
    New TypedValue(1000, "ボルト規格"), _
    New TypedValue(1000, oKW.StringResult), _
    New TypedValue(1000, "単価"), _
    New TypedValue(1070, oPrice), _
    New TypedValue(1002, "}")

oTr.Commit()
```

# 拡張エンティティ データの参照

```
Dim oEnt As Entity
oEnt = oTr.GetObject(oRst.ObjectId, OpenMode.ForRead)

Dim oTbl As RegAppTable
oTbl = oTr.GetObject(oDb.RegAppTableId, OpenMode.ForWrite, False)
If Not oTbl.Has("ST_Bolt") Then
    Dim oRec As RegAppTableRecord
    oRec = New RegAppTableRecord
    oRec.Name = "ST_Bolt"
    oTbl.Add(oRec)
    oTr.AddNewlyCreatedDBObject(oRec, True)
End If

Dim oRB As ResultBuffer = oEnt.XData
If Not IsNothing(oRB) Then
    Dim oWK As TypedValue
    Dim oEnu As IEnumerator = oRB.GetEnumerator()
    While oEnu.MoveNext()
        oWK = CType(oEnu.Current(), TypedValue)
        oWK.GetType.ToString()
        oEd.WriteLine(vbCrLf + oWK.TypeCode.ToString + " = " + oWK.Value.ToString)
    End While
Else
    oEd.WriteLine(vbCrLf + "ボルト規格は付加されていません...")
End If

oTr.Commit()
```

# 拡張エンティティ データの削除

- アプリケーション名を上書き付加することで削除

```
Dim oEnt As Entity
oEnt = oTr.GetObject(oRst.ObjectId, OpenMode.ForWrite)

Dim oTbl As RegAppTable
oTbl = oTr.GetObject(oDb.RegAppTableId, OpenMode.ForWrite, False)
If Not oTbl.Has("ST_Bolt") Then
    Dim oRec As RegAppTableRecord
    oRec = New RegAppTableRecord
    oRec.Name = "ST_Bolt"
    oTbl.Add(oRec)
    oTr.AddNewlyCreatedDBObject(oRec, True)
End If

oEnt.XData = New ResultBuffer(New TypedValue(1001, "ST_Bolt"))

oTr.Commit()
```

# 実際の拡張エンティティ データ

- AutoLISP 式での評価

取得したアプリケーション名を指定  
\* は全アプリケーション名の取得を意味

- `(entget (car (entsel)) (list"*"))`

- 円のエンティティ リスト

コマンド: (entget (car (entsel)) (list"\*"))  
オブジェクトを選択: ((-1 . <図形名: 7ffffcc2800>) (0 . "CIRCLE") (330 . <図形名: 7ffffcc09f0>) (5 . "1B8") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbCircle") (10 1548.54 1129.95 0.0) (40 . 160.569) (210 0.0 0.0 1.0))

- 拡張データが付加された円のエンティティ リスト

コマンド: (entget (car (entsel)) (list"\*"))  
オブジェクトを選択: ((-1 . <図形名: 7ffffcc2800>) (0 . "CIRCLE") (330 . <図形名: 7ffffcc09f0>) (5 . "1B8") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbCircle") (10 1548.54 1129.95 0.0) (40 . 160.569) (210 0.0 0.0 1.0) (-3 ("MYDATA" (1002 . "{") (1070 . 40) (1040 . 162.0) (1040 . 60.0) (1000 . "単価") (1002 . "})))))

アプリケーション名で拡張データを識別

# 実際の拡張エンティティ データ

## ■ 複数の拡張データが付加された円のエンティティ リスト

```
コマンド: (entget (car (entsel)) (list "*"))  
オブジェクトを選択: ((-1 . <図形名: 7ffffcc2800>) (0 . "CIRCLE") (330 . <図形名: 7ffffcc09f0>) (5 . "1B8") (100 . "AcDbEntity") (67 . 0)  
(410 . "Model") (8 . "0") (100 . "AcDbCircle") (10 1634.73 1026.55 0.0) (40 . 223.47) (210 0.0 0.0 1.0) (-3  
("MYDATA" (1002 . "{") (1070 . 40) (1040 . 162.0) (1040 . 60.0) (1000 . "単価")  
(1002 . "}")) ("AppTag" (1002 . "{") (1040 . 123.45) (1070 . 12345) (1000 .  
"文字列") (1002 . "}"))))
```

16 キロバイト以内であれば、拡張データ領域 (-3 以降) に  
複数のアプリケーション名で拡張データを付加することが可能

## ■ AutoCAD 自身も拡張データを利用している

```
コマンド: (entget (car (entsel)) (list "*"))  
オブジェクトを選択: ((-1 . <図形名: 7ffffcc2920>) (0 . "DIMENSION") (330 . <図形名:  
7ffffcc09f0>) (5 . "1CA") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 .  
"0") (100 . "AcDbDimension") (2 . "*D3") (10 1907.3 409.659 0.0) (11 1874.54  
408.357 0.0) (12 0.0 0.0 0.0) (70 . 33) (1 . "") (71 . 5) (72 . 1) (41 . 1.0)  
(42 . 65.4409) (73 . 0) (74 . 0) (75 . 0) (52 . 0.0) (53 . 0.0) (54 . 0.0) (51  
. 0.0) (210 0.0 0.0 1.0) (3 . "ISO-25") (100 . "AcDbAlignedDimension") (13  
1844.2 383.686 0.0) (14 1909.29 390.435 0.0) (15 0.0 0.0 0.0) (16 0.0 0.0 0.0)  
(40 . 0.0) (50 . 0.0) (-3 ("ACAD" (1000 . "DSTYLE") (1002 . "{") (1070 . 176)  
(1070 . 1) (1070 . 177) (1070 . 1) (1070 . 178) (1070 . 3) (1002 . "}"))))
```

上書き（優先）寸法スタイルを参照した  
並行寸法のエンティティ データ

# 拡張エンティティ データ利用上の注意

- 利用するアプリケーション名はユニーク（一意）なものに
    - 他のアプリケーションが利用している名前は避けてください
    - 図形に付加する拡張データの内容はバラバラ
  - アプリケーション名の多用は避けてください
    - 例）図形毎に AppTag1、AppTag2、AppTag3 ... の
    - 特定の拡張データ取得でパフォーマンスに影響します
- 例)

```
(ssget "X" (list (cons -3 (list (list "ACAD")))))
```

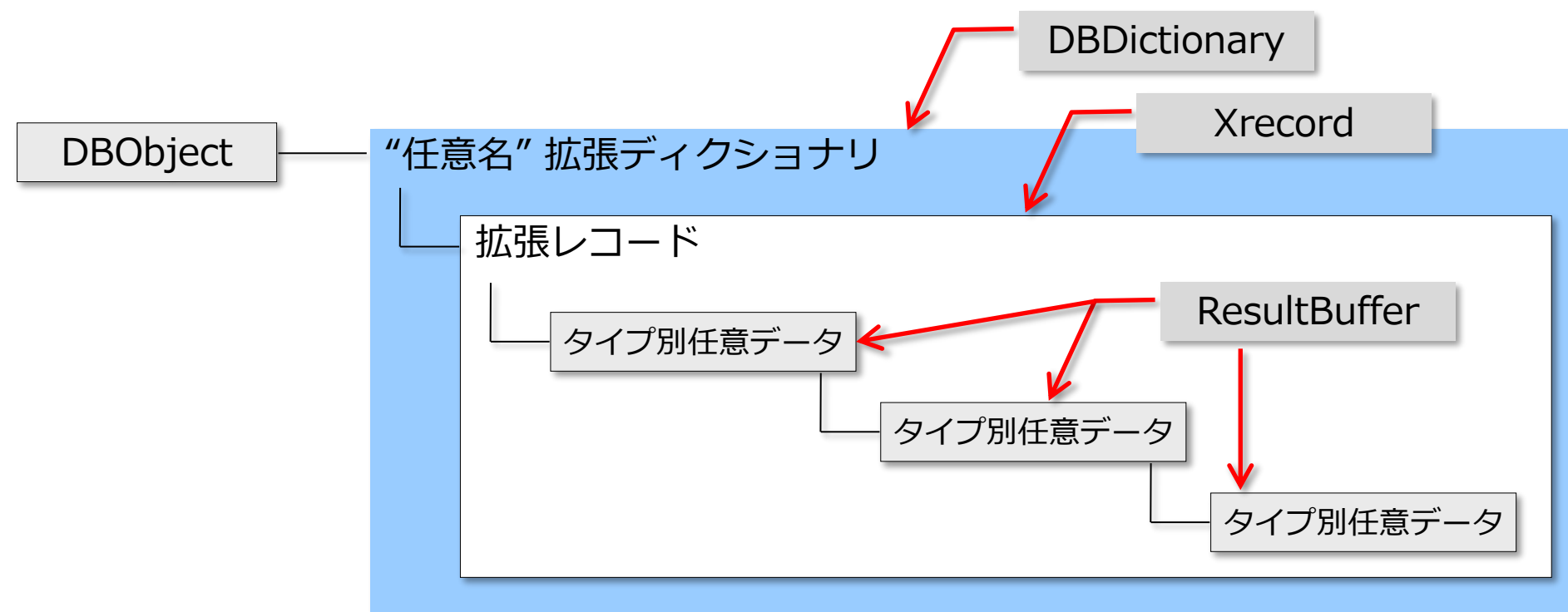
または

```
(ssget "X" '((-3 ("ACAD"))))
```

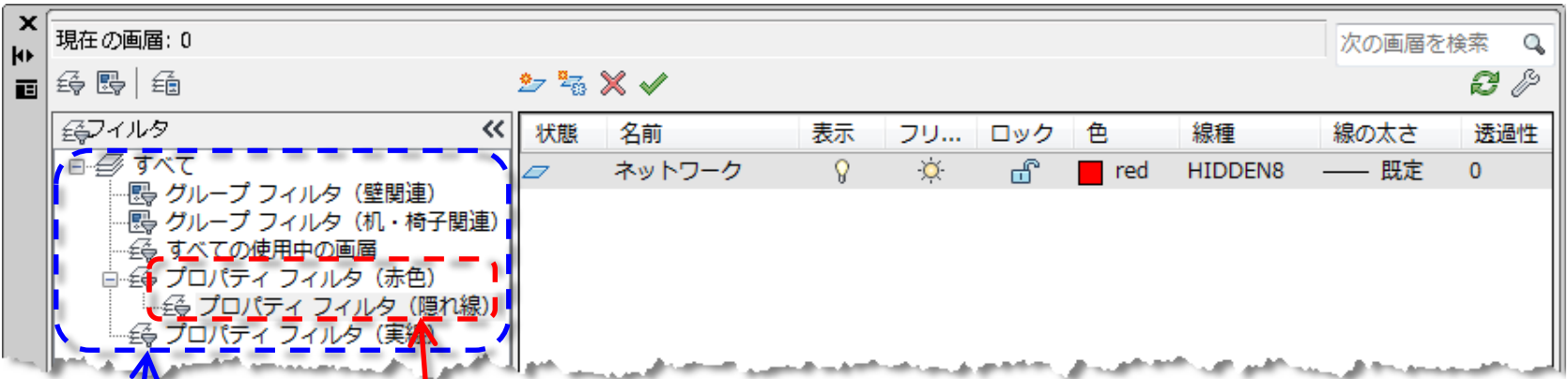
図面全体からアプリケーション名 "MYDATA" の拡張エンティティデータが付加された図形すべてを選択セットで取得

# 拡張ディクショナリとは？

- 任意のオブジェクトに DBDictionary 領域を任意名で設定
- DBDictionary に Xrecord オブジェクトを付加
- Xrecord は DXF グループコードで識別されるタイプ別にチェーン化
- チェイン化とはデータ領域の論理的なつながり

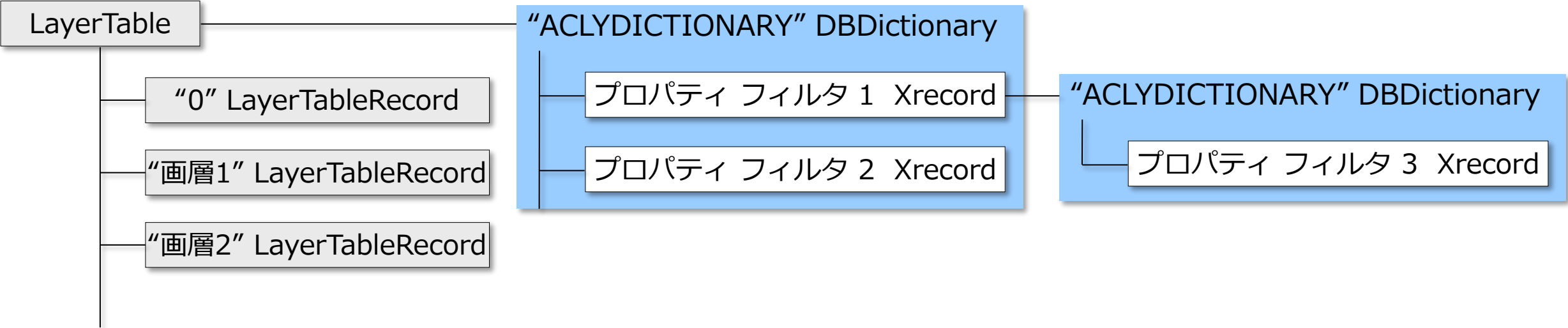


# 拡張ディクショナリの例：画層フィルタ



画層フィルタには、**グループ フィルタ**と**プロパティ フィルタ**の2種類がある

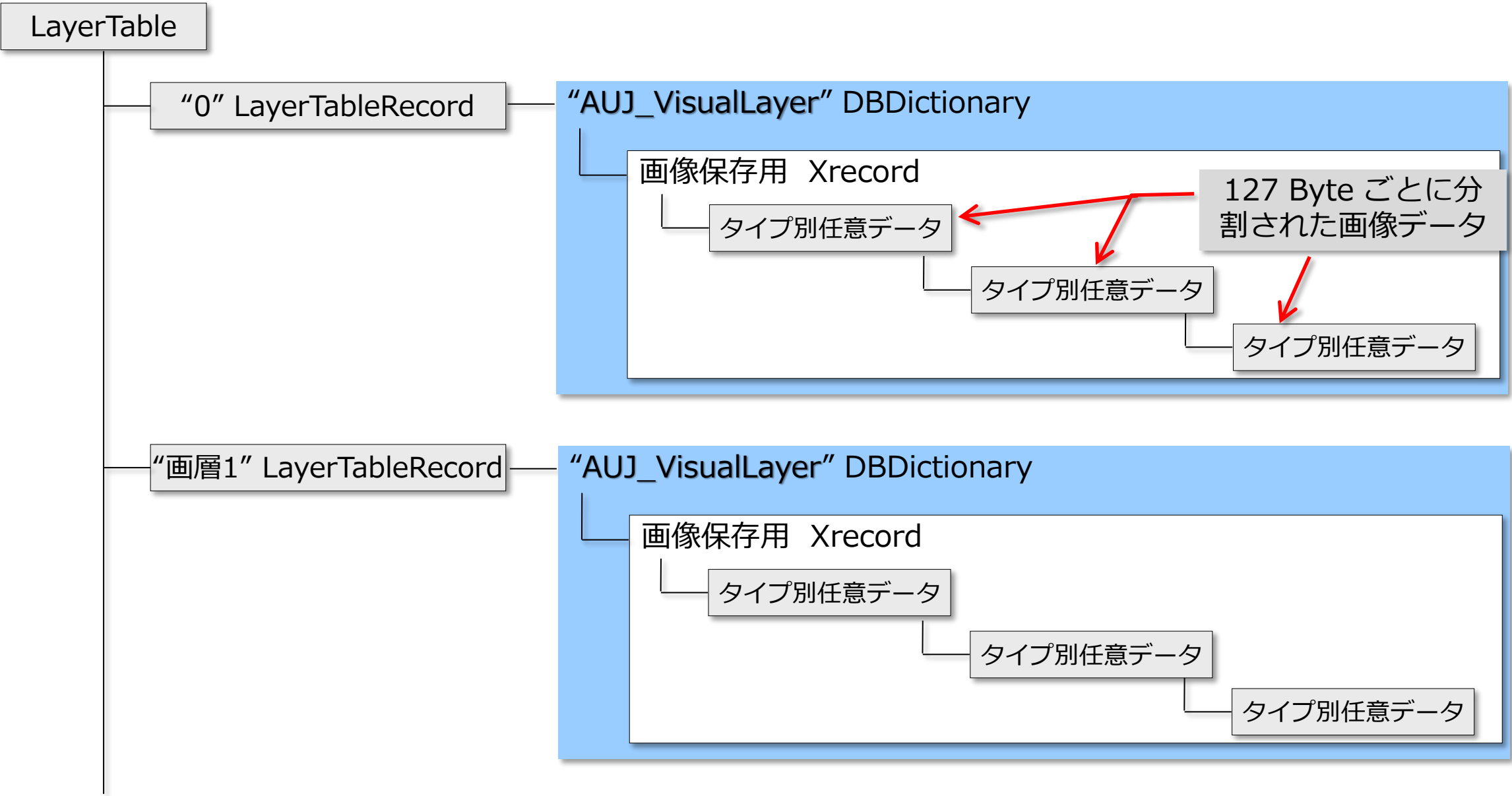
画層フィルタは**階層化**することができる





# 拡張ディクショナリの利用例

- 各画層に画層のキャプチャ画像をバイナリ データとして付加したい !!



# 拡張ディクショナリへのバイナリ データの付加

```
Dim oBitmap As Bitmap = New Bitmap ` キャプチャ画像が生成されていると仮定
Dim oLt As LayerTable = oTr.GetObject(oDb.LayerTableId, OpenMode.ForRead)
Dim oId As ObjectId = oLt.Item("0")
Dim oLtr As LayerTableRecord = oTr.GetObject(oId, OpenMode.ForWrite)
oId = oLtr.ExtensionDictionary
```

“0” 画層の拡張ディクショナリ領域を取得

```
Dim oStream As New MemoryStream()
oBitmap.Save(oStream, ImageFormat.Png)
oStream.Position = 0
```

```
Const kMaxChunkSize As Integer = 127
Dim oRb As ResultBuffer = New ResultBuffer
Dim nIndex As Integer = 0
```

```
While nIndex < oStream.Length
    Dim nLength As Integer = Math.Min(oStream.Length - nIndex, kMaxChunkSize)
    Dim oDataChunk As Byte() = New Byte(nLength - 1) {}
    oStream.Read(oDataChunk, 0, nLength)
    oRb.Add(New TypedValue(DxfCode.ExtendedDataBinaryChunk, oDataChunk))
    nIndex += kMaxChunkSize
End While
```

キャプチャ画像データを 127 Byteごとに分割しながら  
ResultBuffer に格納して連続したチェーンを作成

```
Dim oData As Xrecord = New Xrecord
oData.Data = oRb
```

ResultBuffer チェーンを拡張レコードに格納

```
Dim oDict As DBDictionary = oTr.GetObject(oId, OpenMode.ForWrite)
oDict.SetAt("AUJ_VisualLayer", oData) ` 同一名のディクショナリが存在する場合は自動的に削除される
oTr.AddNewlyCreatedDBObject(oData, True)
oTr.Commit()
```

拡張レコードを AUJ\_VisualLayer 拡張ディクショナリに格納

# 拡張ディクショナリからのバイナリ データの参照

```
Dim oLt As LayerTable = oTr.GetObject(oDb.LayerTableId, OpenMode.ForRead)
Dim oId As ObjectId = oLt.Item("0")
Dim oLtr As LayerTableRecord = oTr.GetObject(oId, OpenMode.ForRead)
oId = oLtr.ExtensionDictionary

Dim oDict As DBDictionary = oTr.GetObject(oId, OpenMode.ForRead)
oId = oDict.GetAt("AUJ_VisualLayer")
If oId.IsNull Then
    oEd.WriteMessage(vbCrLf + "0 画層に AUJ_VisualLayer が付加されていません ...")
End If

Dim oData As Xrecord = oTr.GetObject(oId, OpenMode.ForRead)
Dim oRb As ResultBuffer = oData.Data

Dim oStream As New MemoryStream()
Dim oValues As TypedValue() = oRb.AsArray()
For nIndex As Integer = 0 To oValues.Length - 1
    Dim oDataChunk As Byte() = DirectCast(oValues(nIndex).Value, Byte())
    oStream.Write(oDataChunk, 0, oDataChunk.Length)
Next
oStream.Position = 0
Dim oBitmap As New Bitmap(oStream)
oBitmap.Save("C:¥Restored.png", ImageFormat.Png)
oTr.Commit()
```

“0” 画層の拡張ディクショナリ領域を取得

127 Byte ごとに分割された ResultBuffer チェイン  
を配列として取得してストリームとして結合（復元）

ストリームからのビットマップを PNG ファイルで保存

# 拡張ディクショナリからのバイナリ データの削除

- 拡張ディクショナリを `DBObject.erase()` で削除
  - `DBObject.ReleaseExtensionDictionary` だと  
拡張ディクショナリ領域全体を解放
  - 後で拡張ディクショナリを利用する場合は  
`CreateExtensionDictionary` 呼び出しが必要

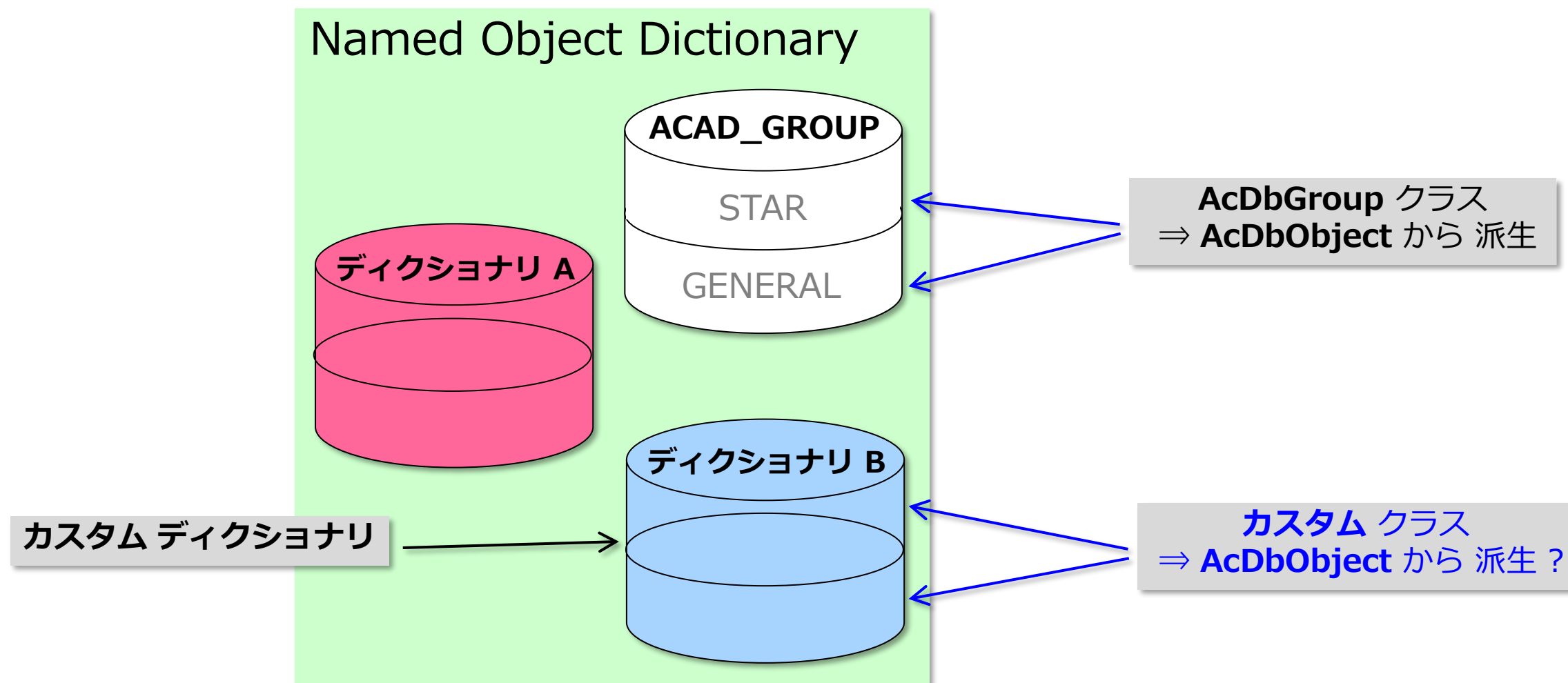
```
Dim oLt As LayerTable = oTr.GetObject(oDb.LayerTableId, OpenMode.ForRead)
Dim oId As ObjectId = oLt.Item("0")
Dim oLtr As LayerTableRecord = oTr.GetObject(oId, OpenMode.ForWrite)
oId = oLtr.ExtensionDictionary

Dim oDict As DBDictionary = oTr.GetObject(oId, OpenMode.ForRead)
oId = oDict.GetAt("AUJ_VisualLayer")
If oId.IsNull Then
    oEd.WriteMessage(vbCrLf + "0 画層に AUJ_VisualLayer が付加されていません ...")
End If

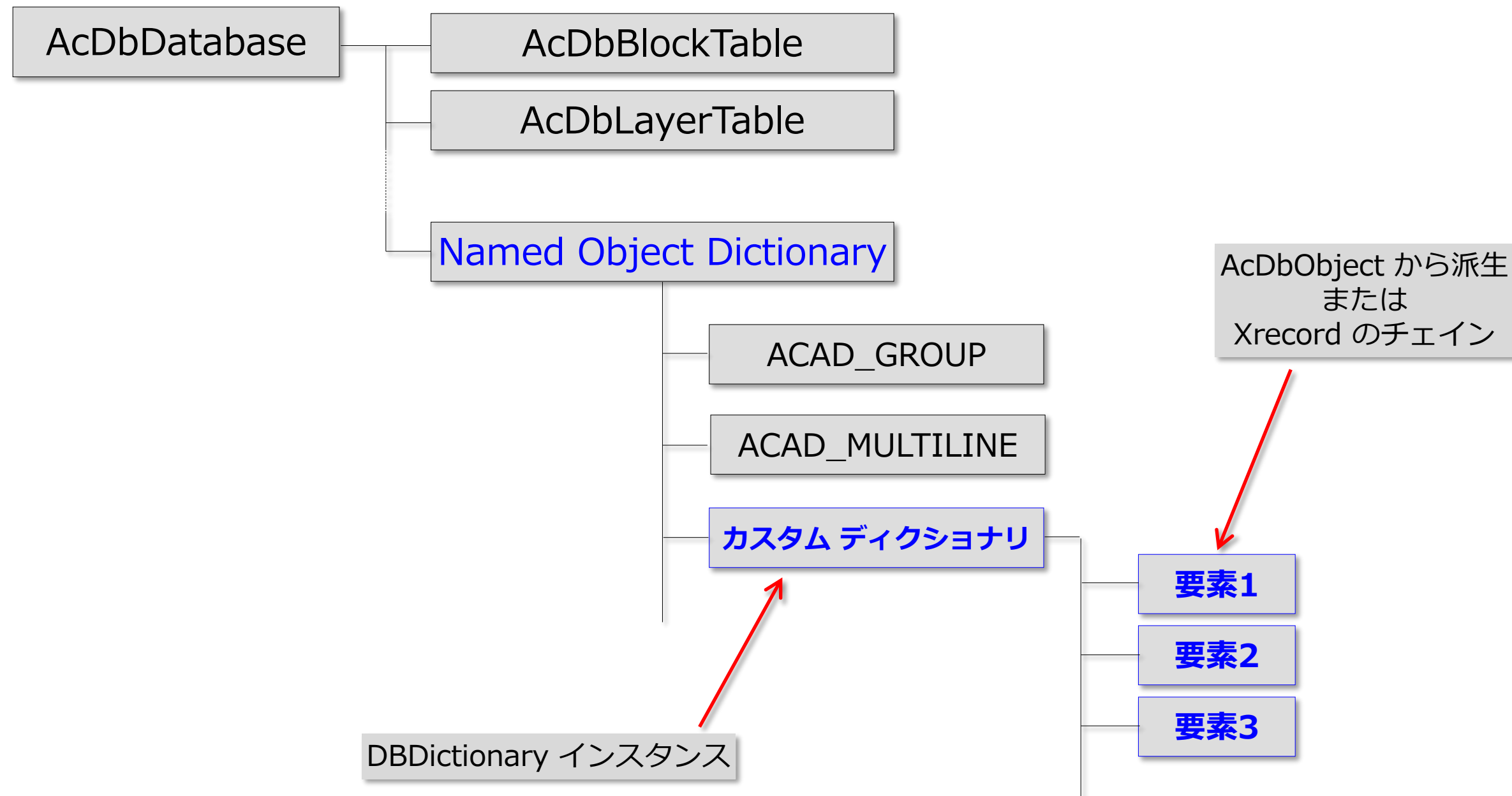
Dim oData As Xrecord = oTr.GetObject(oId, OpenMode.ForWrite)
oData.Erase()
oTr.Commit()
```

# カスタム ディクショナリ

- Named Object Dictionary とディクショナリ 要素
  - 要素には**カスタム オブジェクト**が**拡張ディクショナリ**の付加が可能



# ディクショナリ の構造

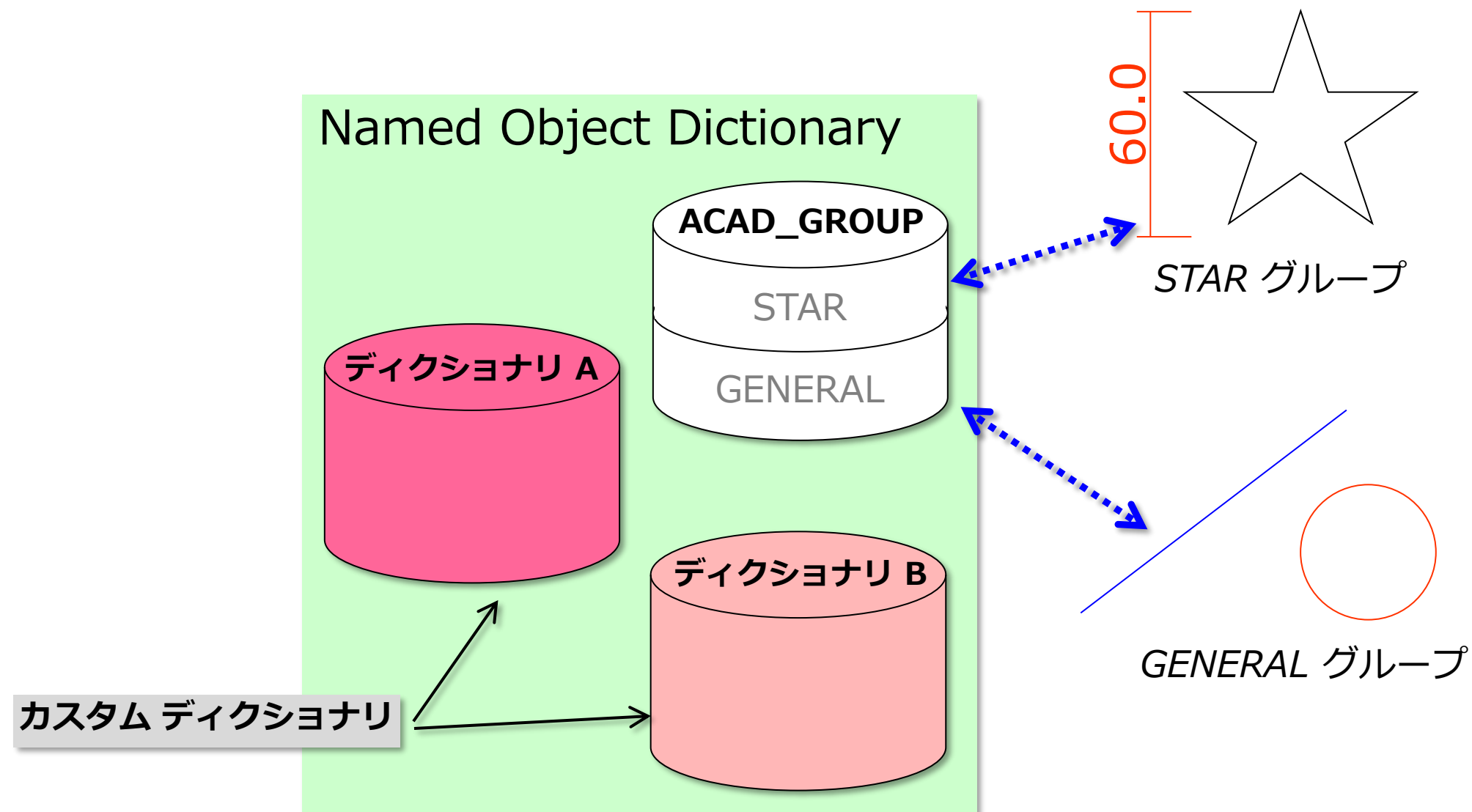


# カスタム デictionary 例

- AutoCAD 組み込み済のdictionary
  - “ACAD\_GROUP”
    - GROUP コマンドがdictionaryを利用
    - 各グループのエンティティ情報を保持
  - “ACAD\_MLINESSTYLE”
    - MLSTYLE コマンドがdictionaryを利用
    - マルチラインスタイル

# カスタム ディクショナリ 例

- “ACAD\_GROUP” ディクショナリ





# Lesson 8: カスタム データ

- 拡張エンティティの付加、参照を実装
  - 付加データの仕様：
    - アプリケーション名（固定）：“トレーニング”
    - 氏名（任意入力）：文字列
    - 身長（任意入力）：実数
    - 年齢（任意入力）：整数
- 実装コマンド
  - 拡張エンティティの付加：[AddXData コマンド](#)
  - 拡張エンティティの参照：[RefXData コマンド](#)

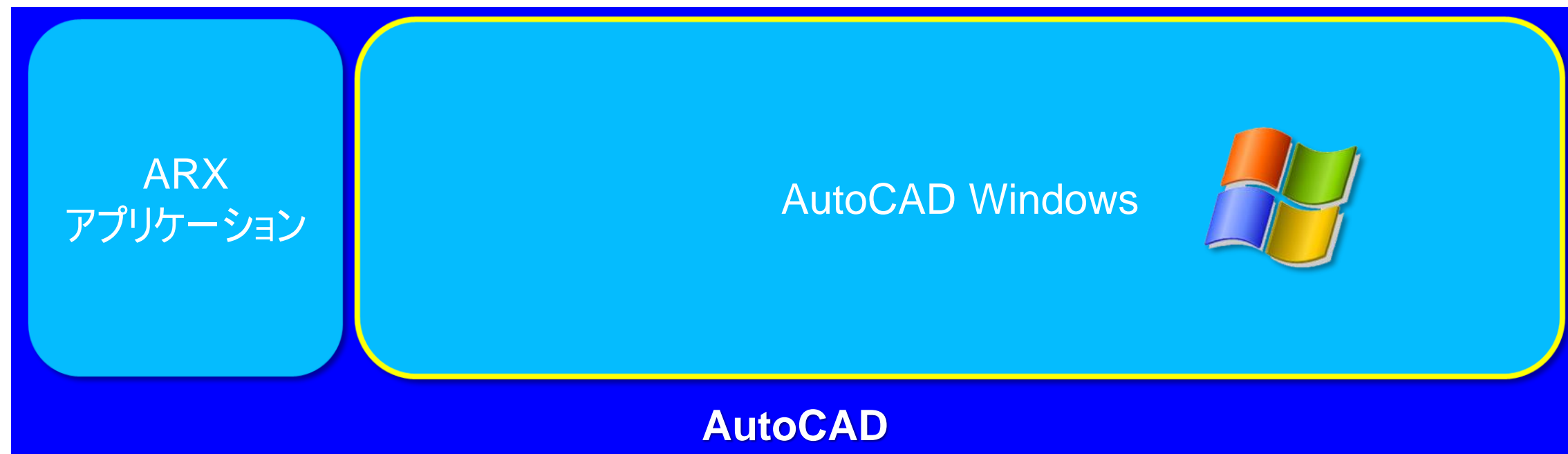
# 参考 Web ページ

- Autodesk Knowledge Network
  - <https://knowledge.autodesk.com/ja/support>
- AutoCAD 開発関連ブログ
  - Through the interface
    - <http://through-the-interface.typepad.com/>
  - AutoCAD DevBlog
    - <http://adndevblog.typepad.com/autocad/>
- 日本語ブログ
  - Technology Perspective from Japan
    - [http://adndevblog.typepad.com/technology\\_perspective/](http://adndevblog.typepad.com/technology_perspective/)
- .NET 言語変換 (VB.NET ⇔ C#)
  - <https://converter.telerik.com/>

# AutoCAD アーキテクチャ変更の影響

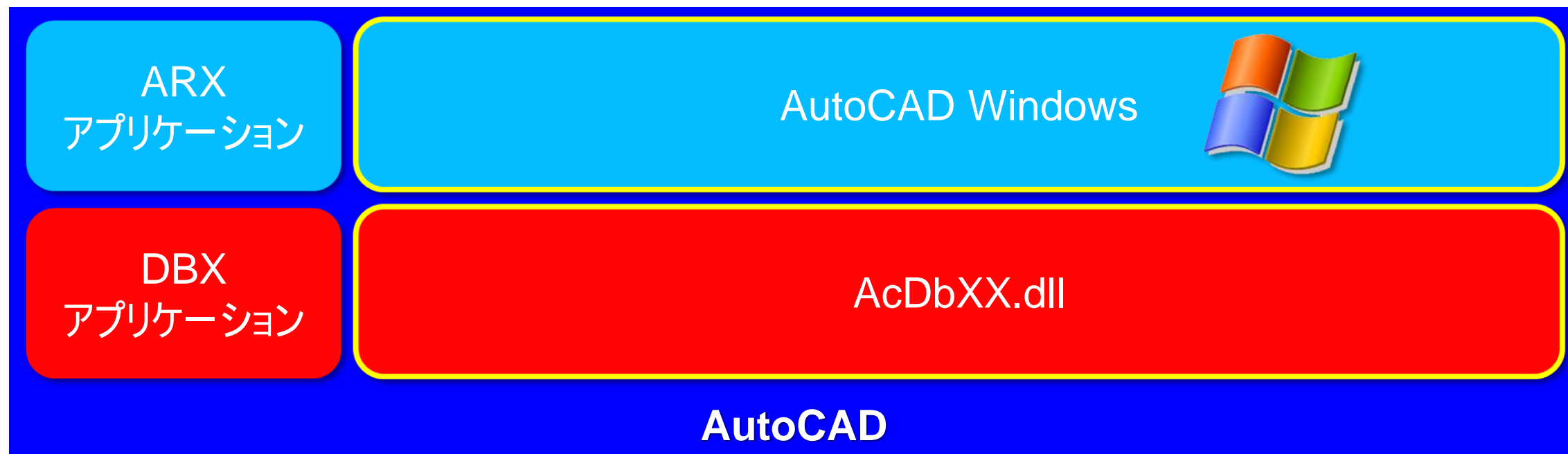
# AutoCAD アーキテクチャ変更の遷移 1

- AutoCAD R12～R14 世代



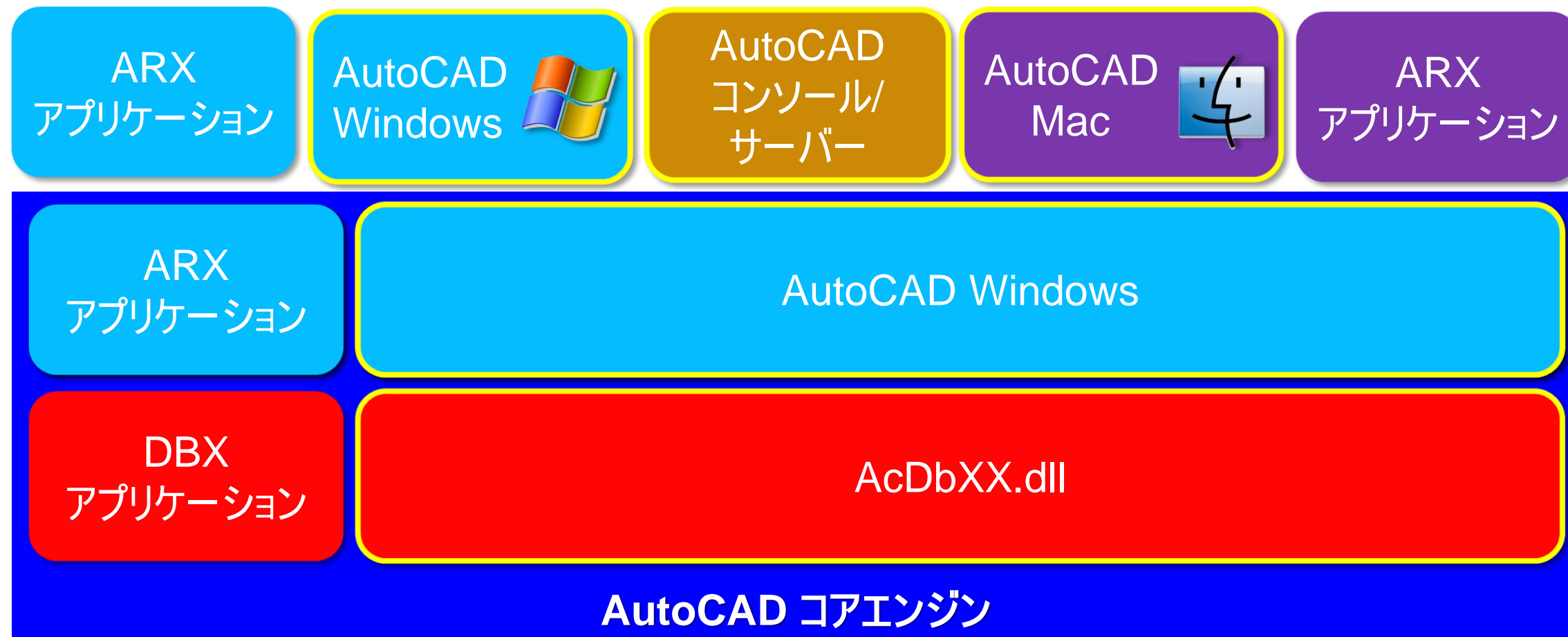
# AutoCAD アーキテクチャ変更の遷移 2

- AutoCAD 2000～2010 世代



# AutoCAD アーキテクチャ変更の遷移 3

- AutoCAD 2014～ （実装開始は AutoCAD 2011 から）



# アーキテクチャ変更の恩恵： AcCoreConsole.exe

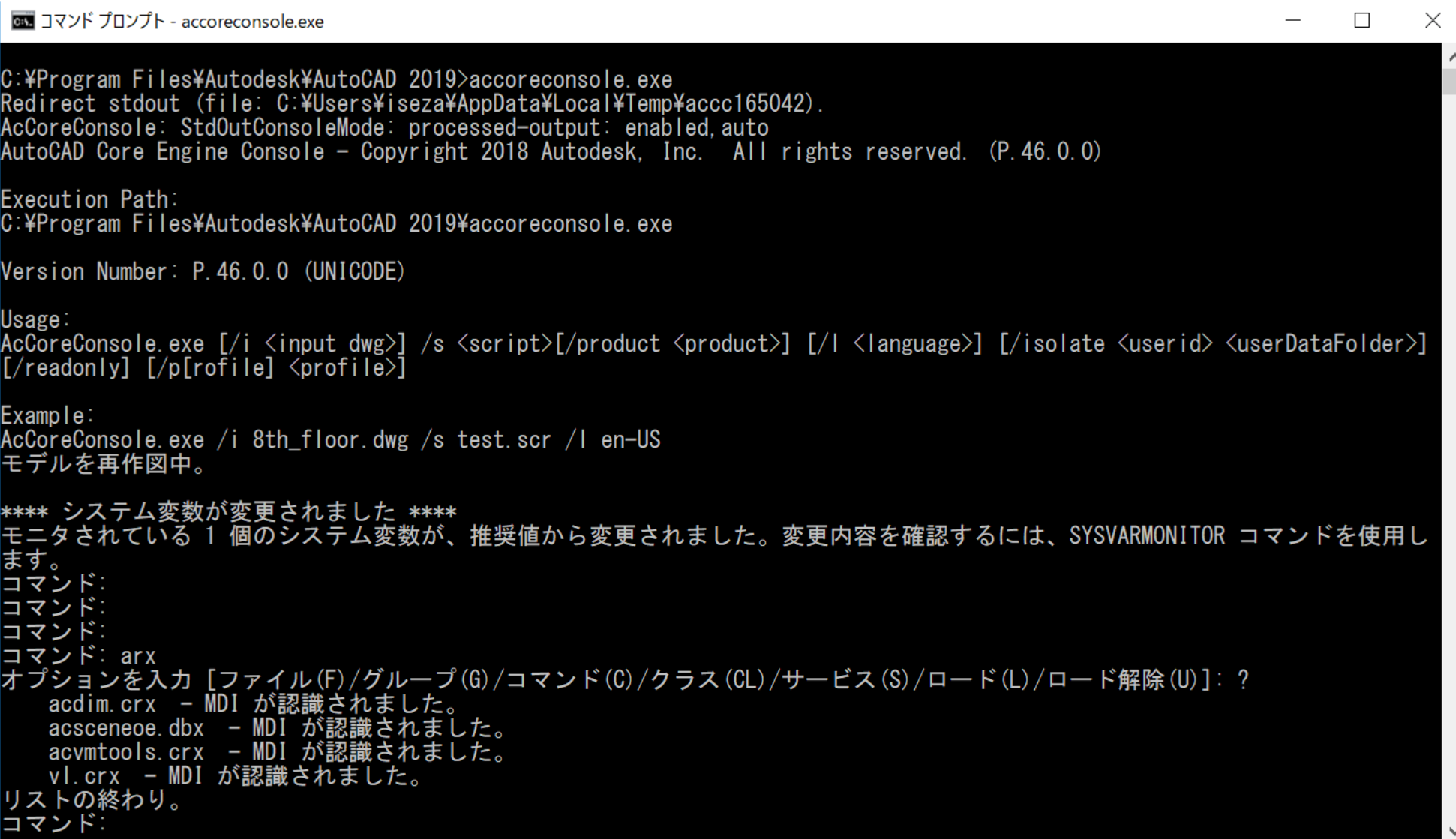
- オーバーヘッドのない AutoCAD

- ユーザ インタフェースなし
- コマンド プロンプトのみ
- バッチ処理に有効

- CRX アドインを認識

- ObjectARX
- .NET API
- AutoLISP

- ソフトウェア使用許諾



```
コマンド プロンプト - accoreconsole.exe
C:\Program Files\Autodesk\AutoCAD 2019>accoreconsole.exe
Redirect stdout (file: C:\Users\iseza\AppData\Local\Temp\accc165042).
AcCoreConsole: StdOutConsoleMode: processed-output: enabled, auto
AutoCAD Core Engine Console - Copyright 2018 Autodesk, Inc. All rights reserved. (P.46.0.0)

Execution Path:
C:\Program Files\Autodesk\AutoCAD 2019\accoreconsole.exe

Version Number: P.46.0.0 (UNICODE)

Usage:
AcCoreConsole.exe [/i <input dwg>] /s <script>[/product <product>] [/l <language>] [/isolate <userid> <userDataFolder>]
[/readonly] [/p[rofile] <profile>]

Example:
AcCoreConsole.exe /i 8th_floor.dwg /s test.scr /l en-US
モデルを再作図中。

**** システム変数が変更されました ****
モニタされている 1 個のシステム変数が、推奨値から変更されました。変更内容を確認するには、SYSVARMONITOR コマンドを使用し
ます。
コマンド:
コマンド:
コマンド:
コマンド: arx
オプションを入力 [ファイル(F)/グループ(G)/コマンド(C)/クラス(CL)/サービス(S)/ロード(L)/ロード解除(U)]: ?
acdim.crx - MDI が認識されました。
acsceneoe.dbx - MDI が認識されました。
acvmttools.crx - MDI が認識されました。
vl.crx - MDI が認識されました。
リストの終わり。
コマンド:
```

# AutoCAD SLA で禁止されている行為・実装

<https://www.autodesk.com/company/terms-of-use/jp/general-terms>

## 15.3 提供物の利用の許容範囲

お客様は、全ての適用法に従ってのみ提供物のアクセス及び利用（並びにアクセス及び利用の許可）を行います（かつ全ての当該適用法に従います）。本規約（追加規約若しくは特別規約を含む）で明示的に許可された場合を除き、又はオートデスクが書面で明示的に別段の許可を行った場合を除き、お客様は、以下の行為を行いません。

- 提供物の全て又は一部の複製、改変、翻案、翻訳、移植、又はこれの二次的著作物の作成、ただし、反対の趣旨の契約上の禁止に関わらず、適用法で明示的に許可された場合を除く。
- 提供物（提供物の機能を含む）の全て若しくは一部の第三者へのサブライセンスの許諾、配布、送信、販売、賃貸、貸付、若しくはその他の方法で利用可能とすること、又は（サービスビューロベースその他による）第三者への提供物の機能の提供
- インターネット、広域ネットワーク（WAN）、その他のローカルでないネットワーク、仮想プライベートネットワーク（VPN）、アプリケーション仮想化技術、リモート仮想化技術、ウェブホスティング、タイムシェアリング、サービスとしてのソフトウェア、サービスとしてのプラットフォーム、サービスとしてのインフラ、クラウドその他のウェブベース、ホスト型等のサービス上又はこれらを通じた提供物のアクセス若しくは利用（オートデスクによるインターネットを通じた提供を除く）

さらに、お客様は以下の行為を行いません。

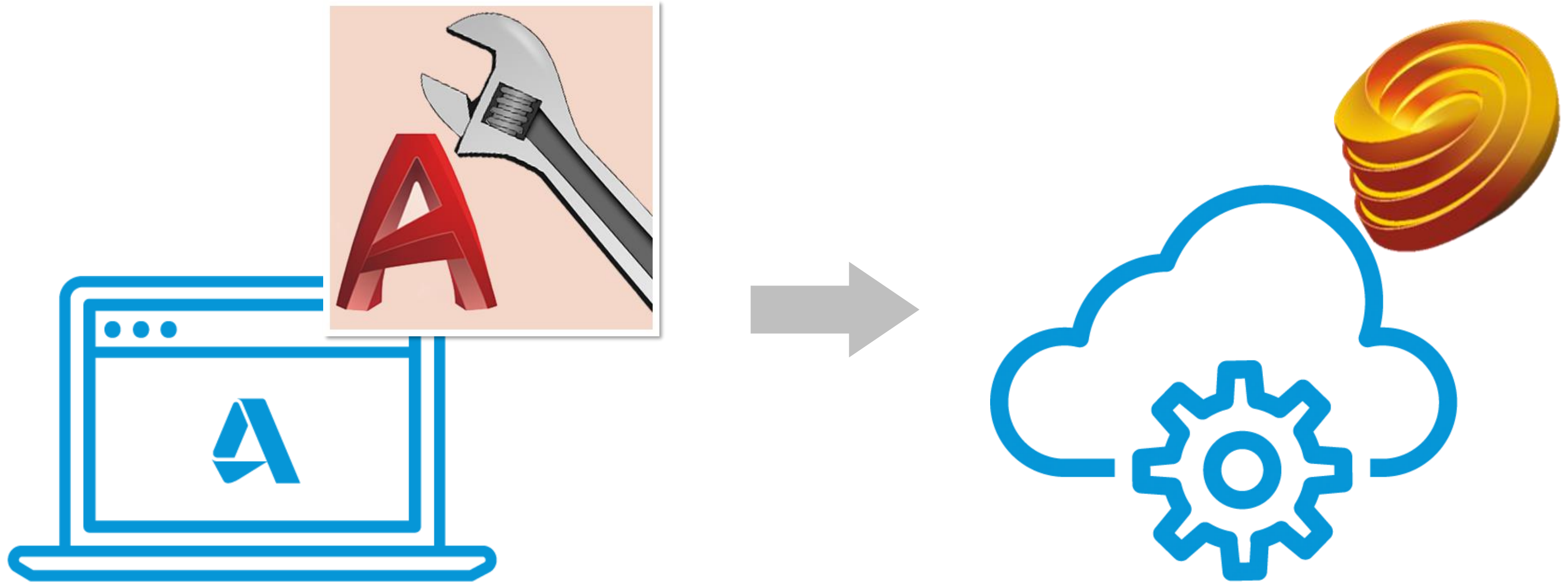
<中略>

- 提供物を「遠隔ロード」のための記憶装置として又はその他のウェブページ若しくはインターネットリソースへの「入り口」若しくは「道標」として使用すること（提供物が提供されるサイト内であるか又はサイトの域内を越えるかを問わない）

<中略>



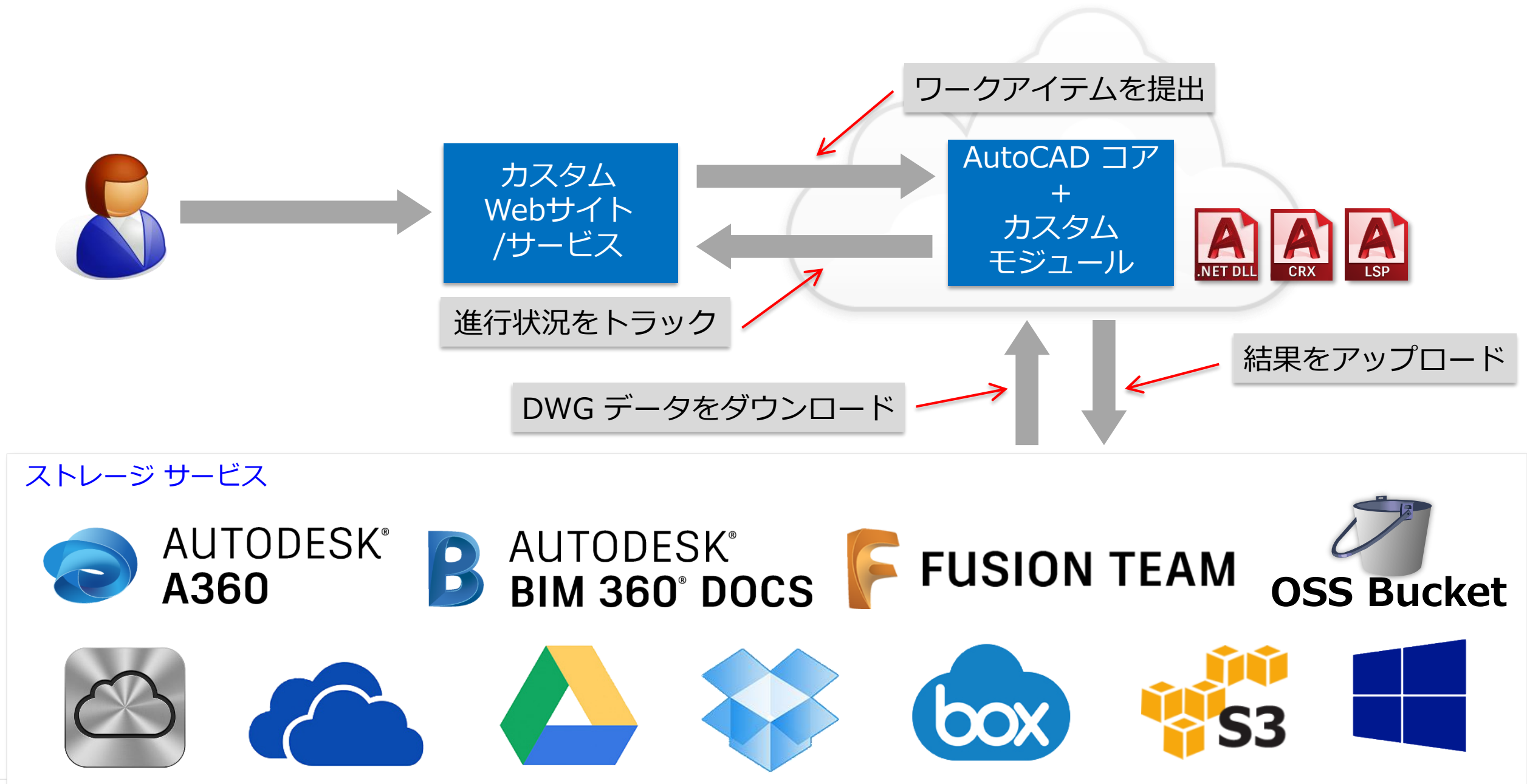
共有サーバー/クラウドに独自にホストする  
AutoCAD を使った不特定多数へのサービス提供は  
**Forge Design Automation API for AutoCAD** で



# Design Automation API のアーキテクチャ



- クラウドで AcCoreConsole による DWG 処理を提供



# Design Automation API のゴールではないもの

- エンドユーザ向けのサービスでは ありません
  - 開発者向けのサービスです
- DWG の対話的な表示/編集では ありません
  - この目的は AutoCAD モバイル アプリ で提供
- サーバー モジュールでは ありません
  - オンプレミス(プライベート)サーバーへのインストール不可



Autodesk is a registered trademark of Autodesk, Inc., and/or its subsidiaries and/or affiliates in the USA and/or other countries. All other brand names, product names, or trademarks belong to their respective holders. Autodesk reserves the right to alter product and services offerings, and specifications and pricing at any time without notice, and is not responsible for typographical or graphical errors that may appear in this document.