

::ISUG

techcast series

PowerBuilder 11: Deploying Business Logic as .NET Assemblies

November 13, 2007

SYBASE®

::ISUG

techcast series

PowerBuilder 11: Deploying Business Logic as .NET Assemblies



Mike Harrold,
Executive Director
International Sybase Users Group

SYBASE®

::ISUG

techcast series

PowerBuilder 11: Deploying Business Logic as .NET Assemblies



Jim O'Neil
Senior Systems Consultant
Sybase, Inc.

SYBASE[®]

- **PowerBuilder roadmap**
- **PowerBuilder .NET support**
- **The “five Ds” of PowerBuilder .NET assemblies**
 - Design the business logic
 - Define the deployment properties
 - Deploy the NVO
 - Develop the calling application
 - Debug the assembly
- **Demo: “the sixth D”**
- **Learn more about it!**

PowerBuilder Roadmap

- **Version 11 released in June 2007**
- **Delivers on third-phase of PowerBuilder's .NET initiative**
 - .NET Windows Forms Deployment (and Smart Client)
 - .NET Web Forms Deployment
 - .NET Assemblies
 - .NET Web Services



PowerBuilder Roadmap

2007 PowerBuilder 11.1

- .NET incremental rebuild
- Informix 10 support
- Vista support
- RadControls for Web Forms
- Miscellaneous fixes

2008 PowerBuilder 11.2

- AJAX support for Web Forms Applications
- EAServer .NET Client Support
- Miscellaneous fixes

2008 PowerBuilder 11.5

- Core .NET enhancements
- Native DataWindow updates
- Updated database support
- Miscellaneous fixes

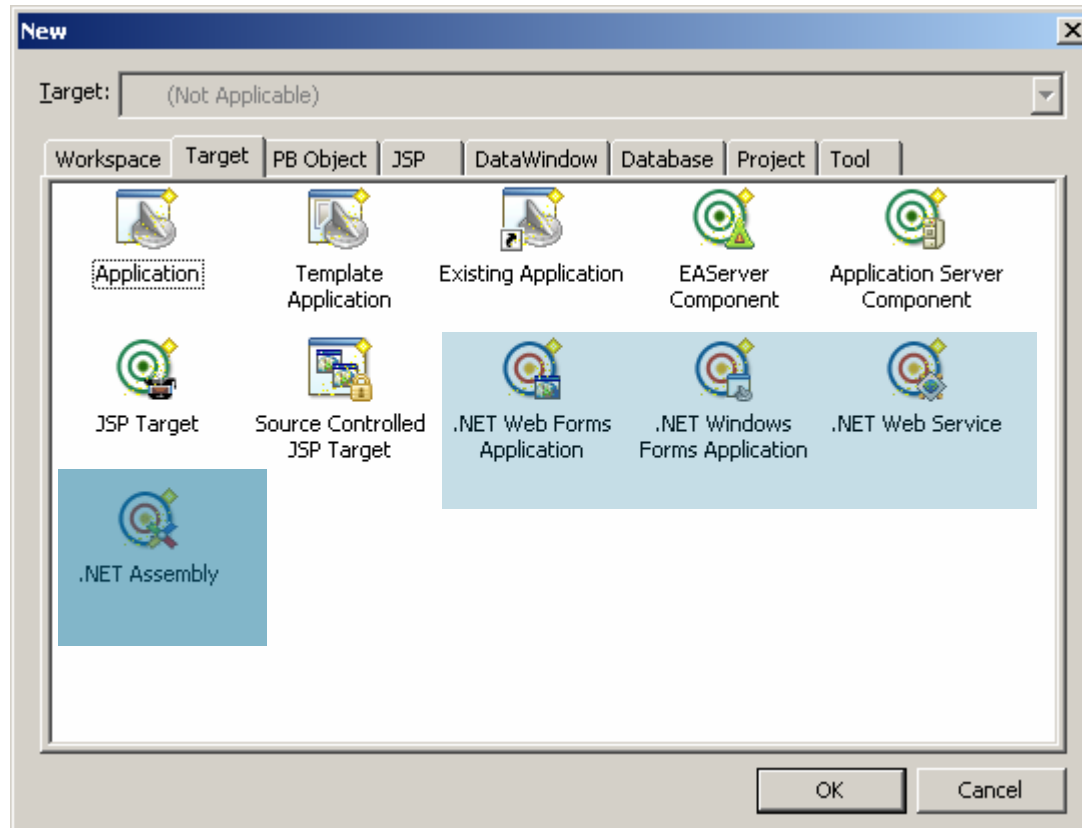
2009 PowerBuilder 12

- .NET in IDE
- WPF/WCF support at design and run-time
- Fully managed code at deployment
- Complete .NET interoperability



PowerBuilder 11 .NET Support

- **PowerBuilder .NET deployment options**



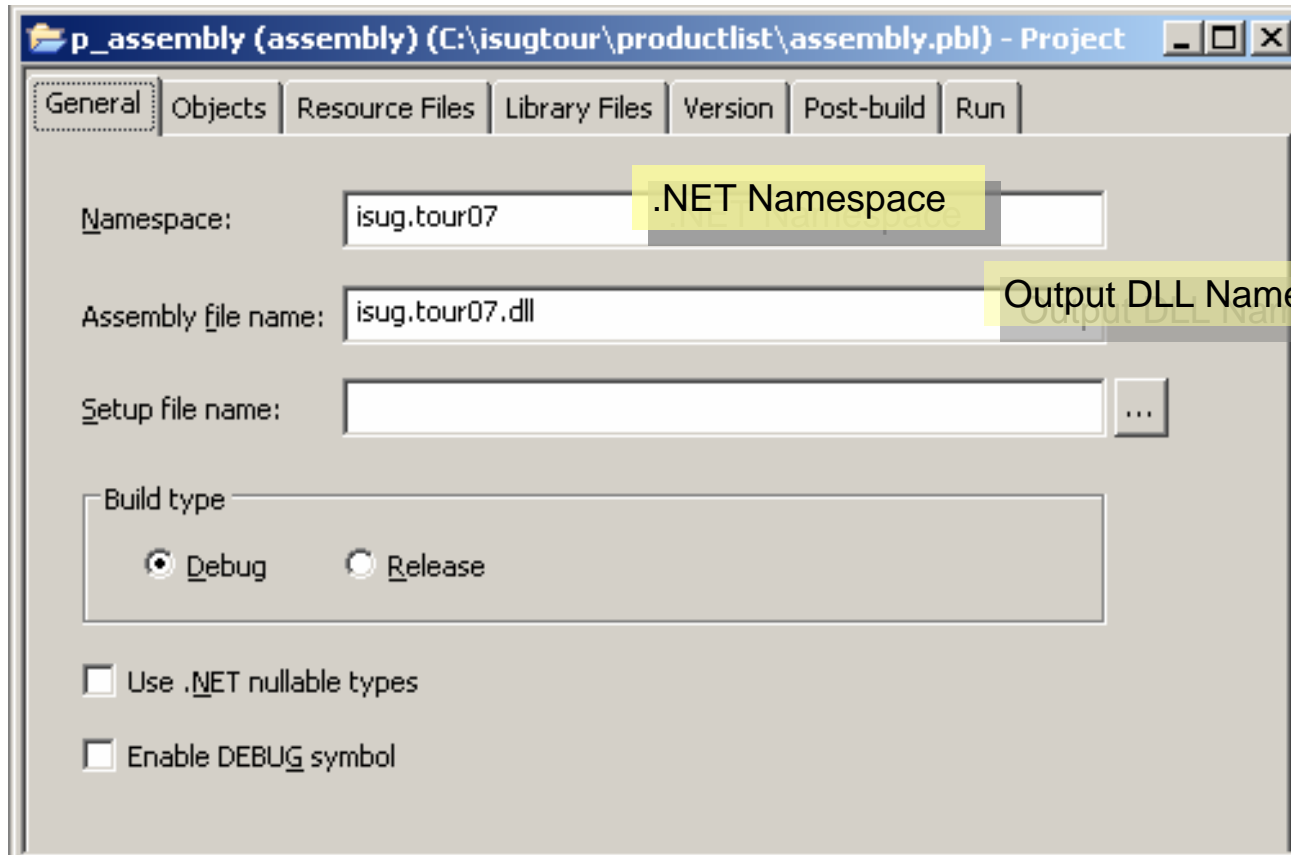
.NET Assembly Definition

- **Building block of .NET applications**
 - Logical unit of functionality (a DLL in physical terms)
 - Implementation independent
 - Contains code – Microsoft Intermediate Language (MSIL) – executed by Common Language Runtime (CLR)
- **Imposes boundaries for contexts of**
 - Security
 - Type
 - Reference scope
 - Versioning
 - Deployment
- **Implications for PowerBuilder developers**
 - Immediate reuse of existing code in .NET architectures
 - “Full citizenship” in .NET application development circles

- **Design the business logic**
 - PowerBuilder 11 supports the deployment of non-visual objects (Custom Class User Objects) as .NET assemblies
 - Existing NVOs can be leveraged by creating a new .NET Assembly Target that references the original target
 - Implement methods in PowerScript as you would for a traditional PowerBuilder application
 - Only a few caveats
 - **Proprietary PowerBuilder types (e.g., datastore, transaction, etc.) cannot be used in public method signatures**
 - **Instance variables are not exposed – define your own getter/setter methods**
 - **Some PowerScript constructs are not supported in .NET deployments**

PowerBuilder .NET Assemblies

- **Define the deployment properties (General)**
 - PowerBuilder project object encapsulates the details



- Define the deployment properties (Objects)

The screenshot shows the PowerBuilder Objects tab for a .NET assembly. The window title is "p_assembly (assembly) (C:\isugtour\productlist\assembly.pbl) - Project". The "Objects" tab is selected, showing a tree view of "Custom class objects" with "assembly.pbl" and "n_prodlst".

Annotations in yellow boxes point to various fields:

- .NET class name** points to the "Object name" field, which contains "n_prodlst".
- .NET method** points to the "Method Name" column in the table, specifically to the "retrieveProducts" entry.
- NVO method** points to the "Function Prototype" column in the table, specifically to the prototype "of_getproductlist (ref s_prodlstitem thelist[], ref".
- Intellisense properties** points to the "Description" text area, which contains "Inventory object".
- NVO list** points to the tree view on the left.

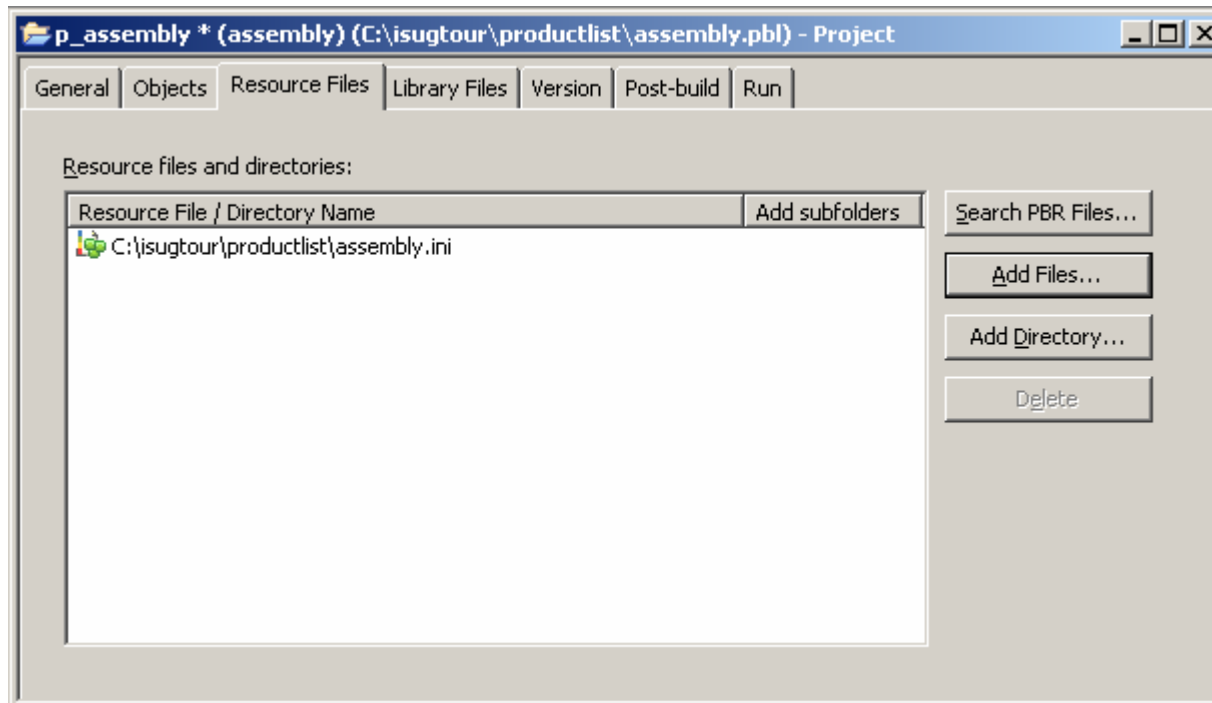
The "Description" field contains the text "Inventory object".

The "Namespace" is "isug.tour07".

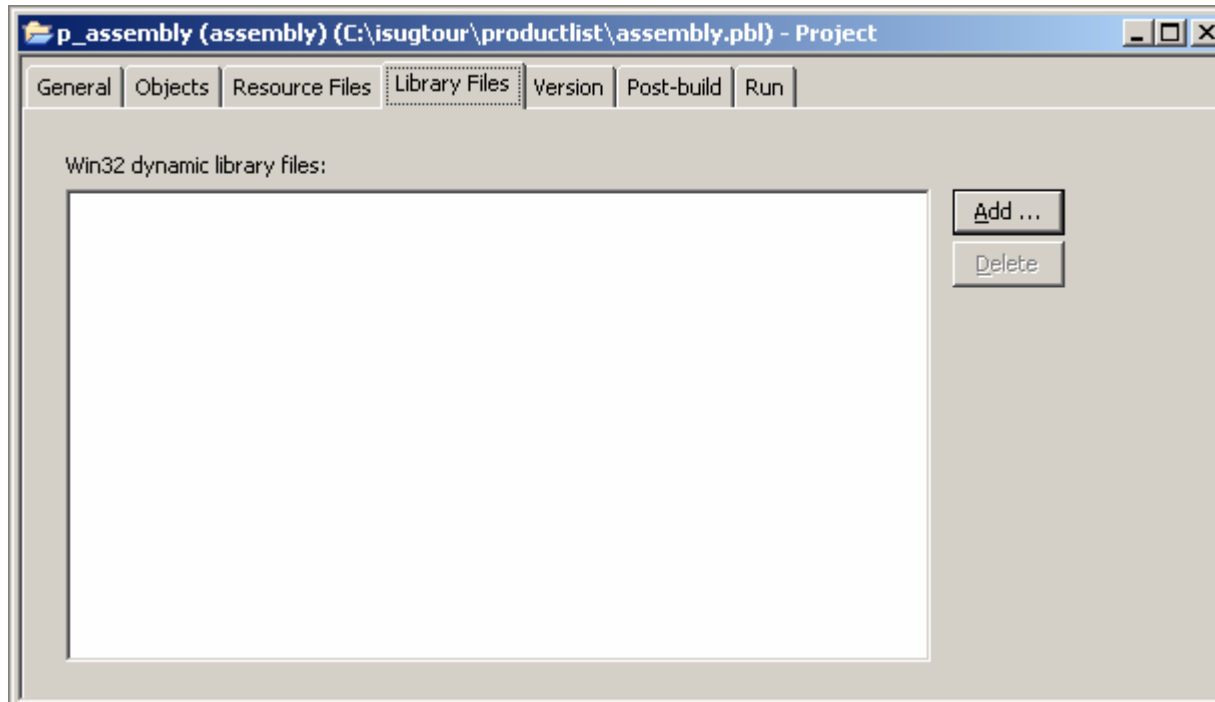
Method Name	Description	Function Prototype
<input checked="" type="checkbox"/> retrieveProducts	Retrieve inventory data	of_getproductlist (ref s_prodlstitem thelist[], ref

Buttons on the right side include: Change Name, Change Description, Select All, and Unselect All.

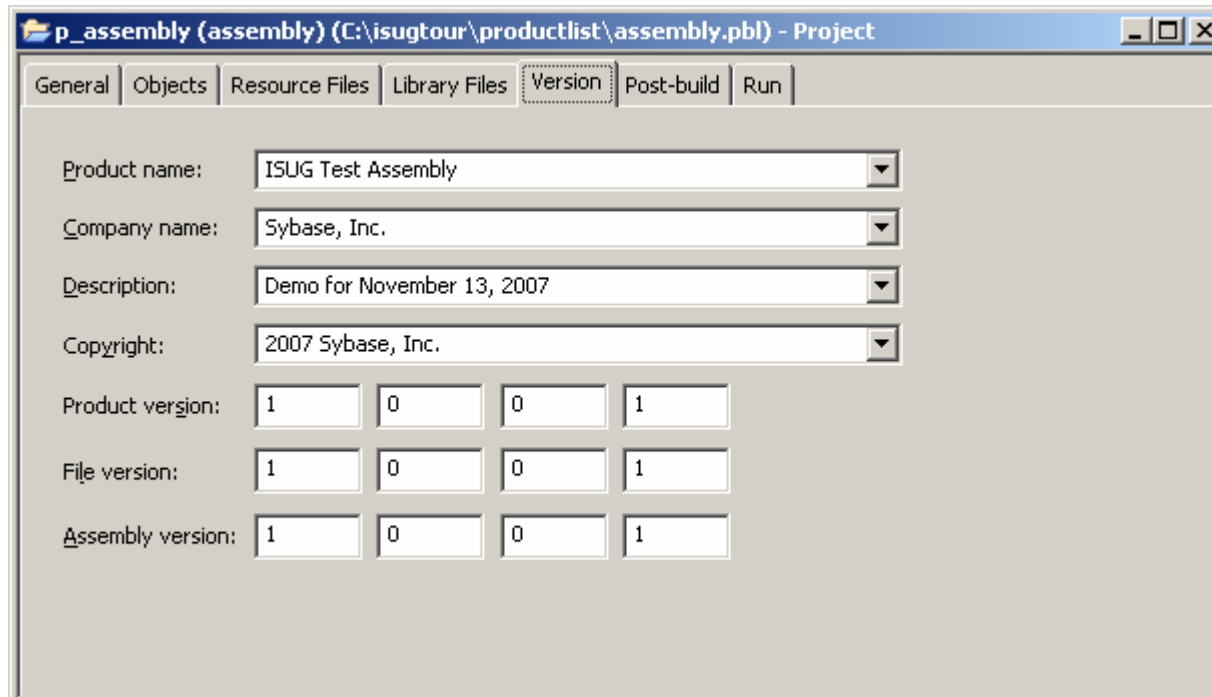
- **Define the deployment properties (Resource Files)**
 - .INI files
 - Text files
 - Images



- **Define the deployment properties (Library Files)**
 - Invocation of most types of native calls (local external functions) is supported
 - List is initially populated by wizard based on introspection of code in the user object

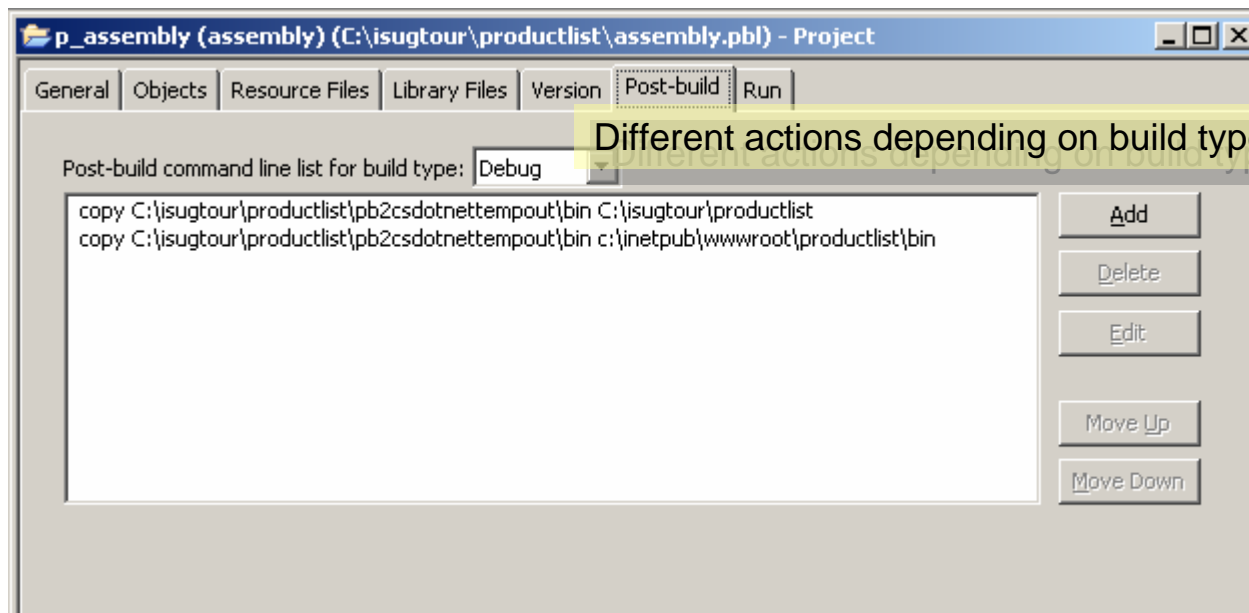


- Define the deployment properties (Version)

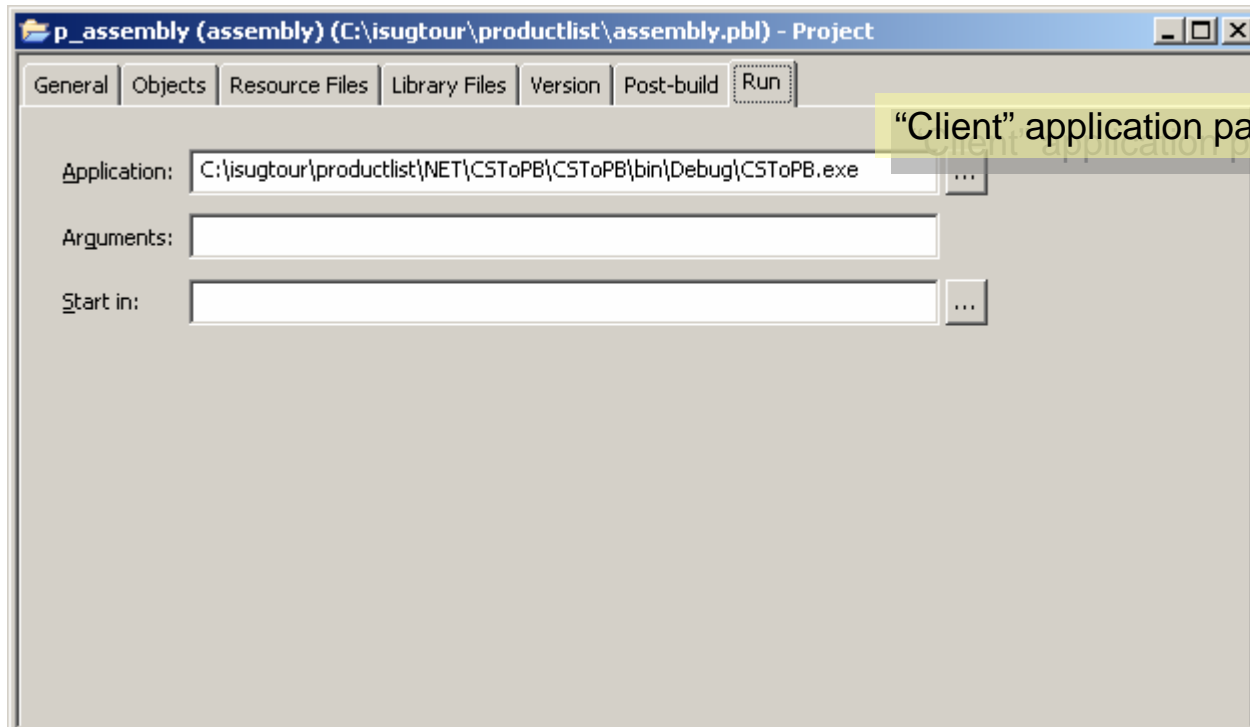


PowerBuilder .NET Assemblies

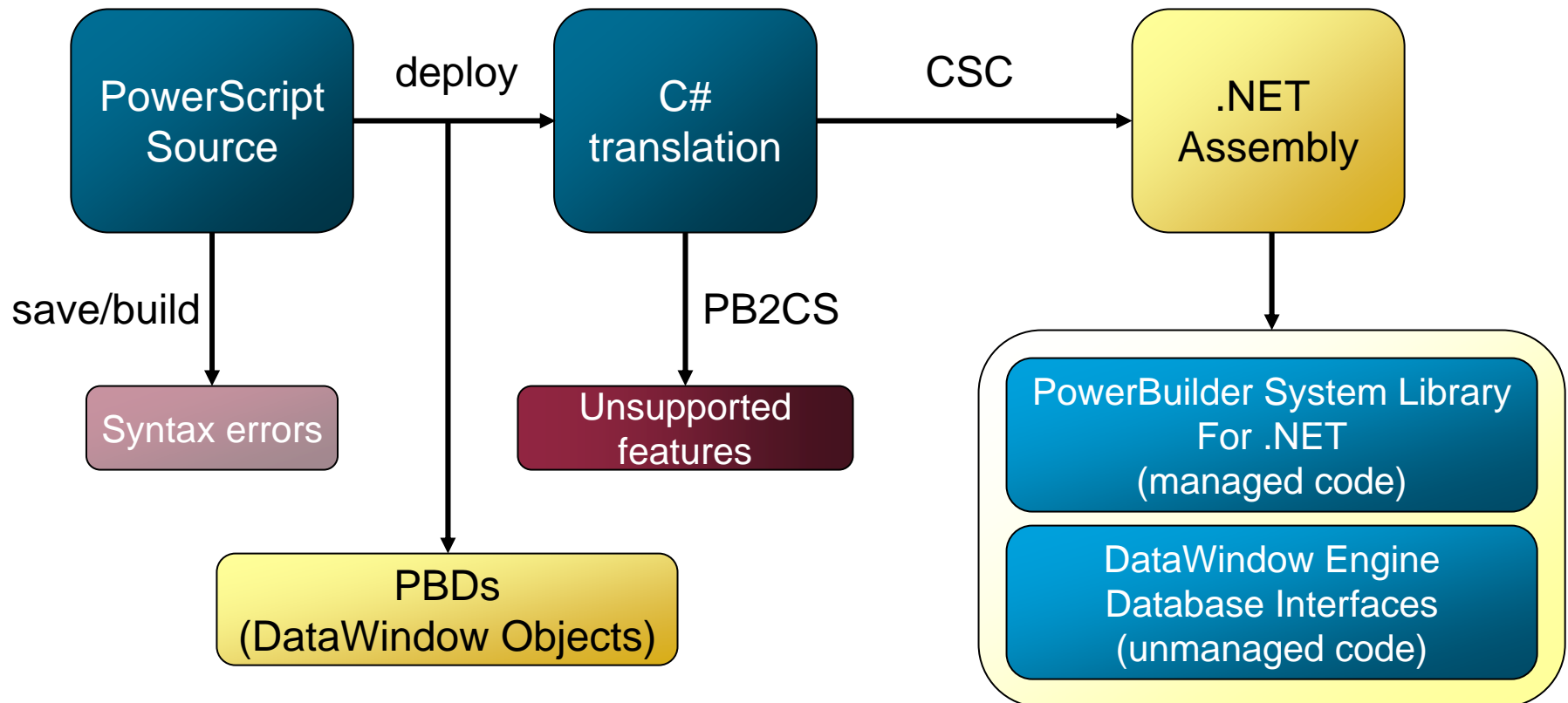
- **Define the deployment properties (Post-build)**
 - Any command recognizable by operating system
 - Sample uses
 - **Invoke .NET obfuscator tool**
 - **Copy output to staging area**
 - **Send page/e-mail indicating build is complete**



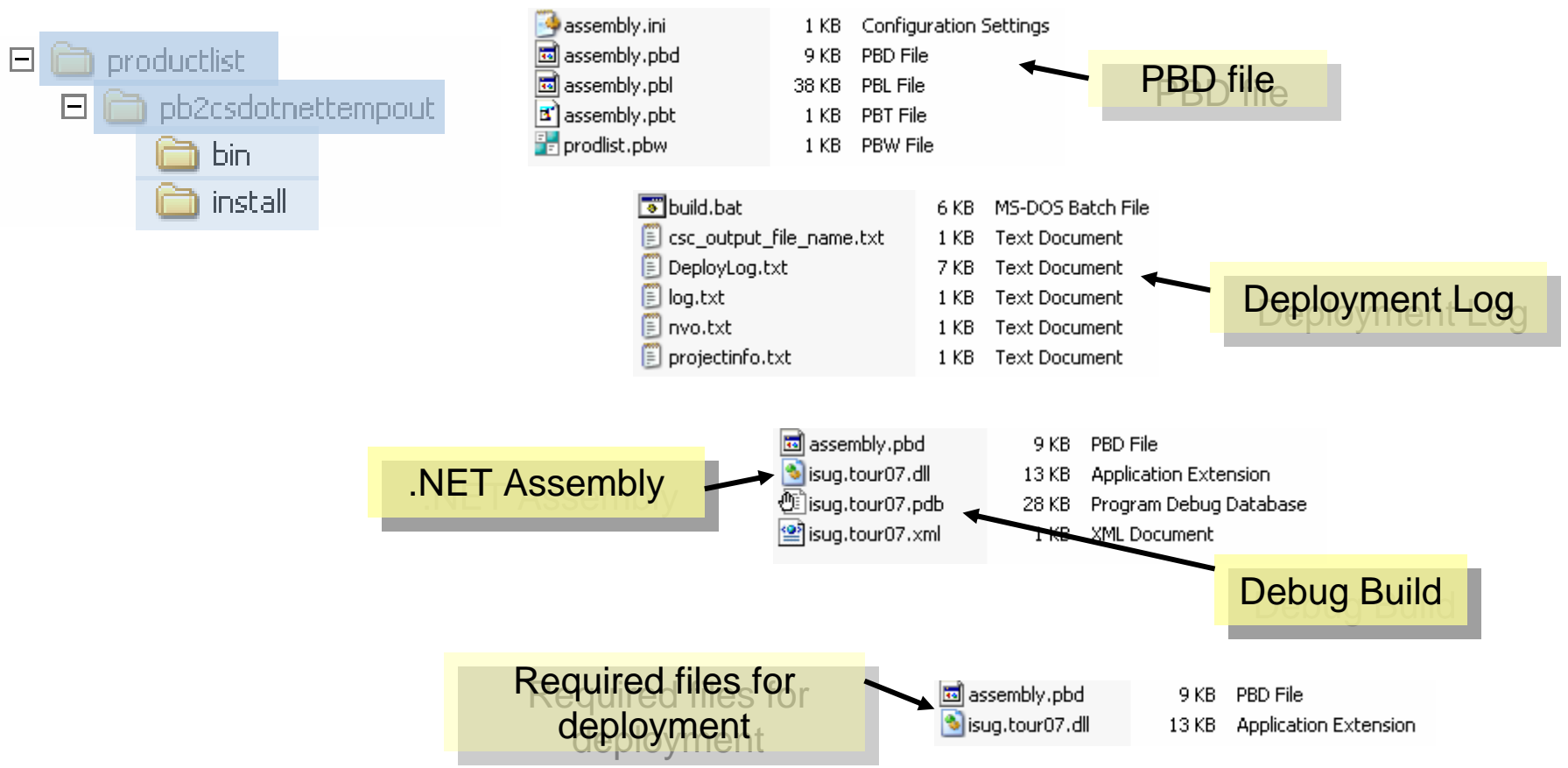
- **Define the deployment properties (Run)**
 - Specify application used to invoke assembly
 - “Running Man”
 - Debugger



- **Deploy the NVO**
 - Use project painter or context menu on target/project to initiate deployment



- **Post-deployment directory structure**



- **Deployment to target environment**

Runtime packager

- **System Library for .NET**
- **Other required files dependent on functionality**



XCopy from /install directory

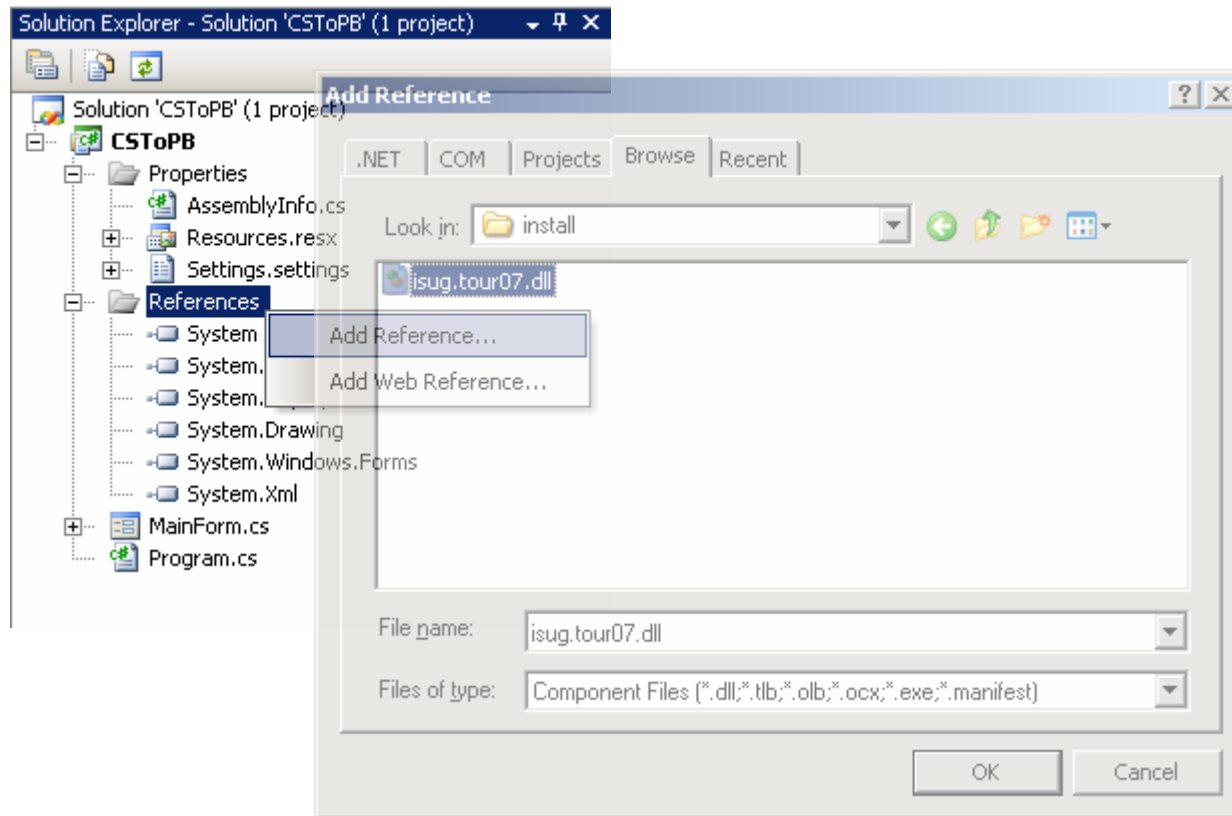
OR

Setup file (MSI) from project object properties

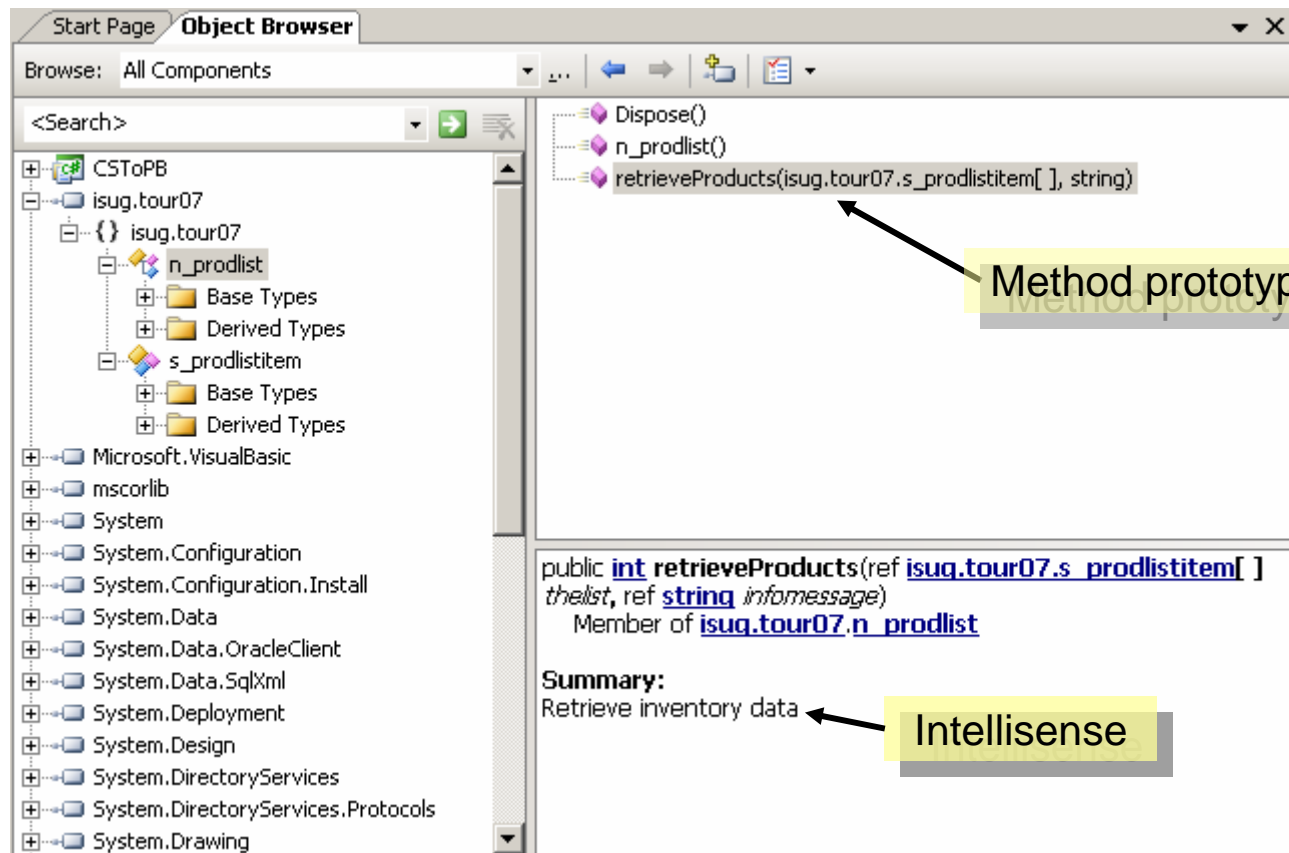


PowerBuilder .NET Assemblies

- **Develop the calling application (Visual Studio .NET)**
 - Add reference to assembly



- Object Browser view



- **Sample invocation using C#**

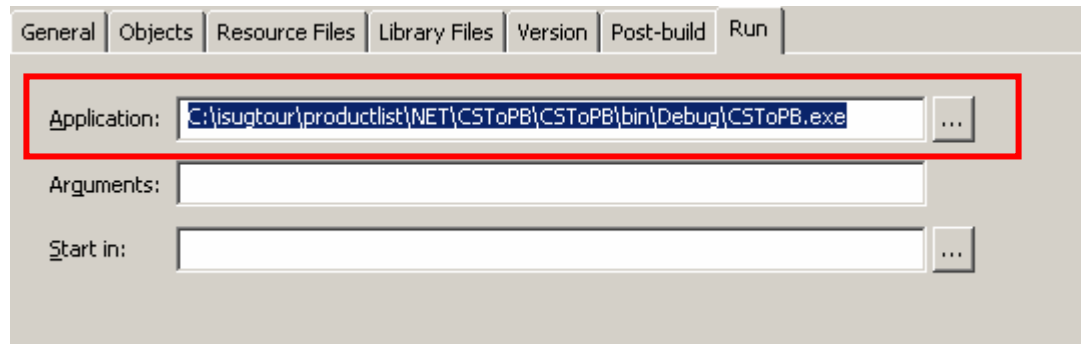
```
private void cb_Populate_Click(object sender, EventArgs e)
{
    isug.tour07.n_prodlist productList;
    isug.tour07.s_prodlistitem[] products = null;

    String infoMsg = null;
    Int32 numItems;

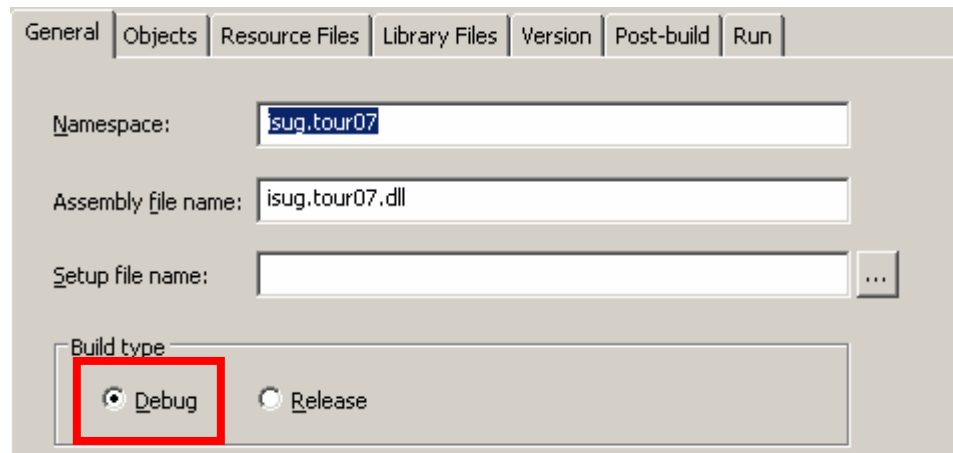
    productList = new n_prodlist();
    numItems = productList.retrieveProducts(ref products,
        ref infoMsg);


    items = new System.Collections.ArrayList();
    foreach (s_prodlistitem t in products)
        items.Add(new Item(t));
    dgrProducts.DataSource = items;
}
```

- **Debug project settings**
 - Specify test application on Run tab



- Specify “Debug” as build type



- **Assembly debugging sequence**
 1. Set breakpoints in PowerBuilder script view
 2. Launch “Run” application using Debug option 
 3. Execution pauses when client application invokes the PowerBuilder assembly and reaches a defined breakpoint
 4. At this point, most “classic PowerBuilder” debugging options are available

HELP! My breakpoints aren't being triggered!

Ensure the correct .PDB file, generated during deployment, is available (typically placed in same directory as assembly)



Learn More About It!

- **ISUG “Leveraging PowerBuilder in the .NET World”**
 - Day-long, hands-on training event
 - Multiple cities in North America through December
 - Visit <http://www.isug.com/pb11tour> for details
- **Sybase webcasts**
 - Six sessions over next few months
 - Focused on new .NET features
 - **PowerBuilder 11 Overview (Nov 20th)**
 - **Windows Forms and .NET Interoperability (Dec 4th)**
 - **Web Forms (Dec 11th)**
 - **Smart Client (Jan 15th)**
 - **Web Services Deployment (Jan 29th)**
 - **Web Services DataWindow (Feb 5th)**
- **PowerBuilder Videos:**
<http://www.sybase.com/products/development/powerbuilder/videos>