CA Gen

Distributed Processing - .NET Server User Guide

Release 8.5

Second Edition

CA Technologies Product References

This document references the following CA Technologies products:

- CA Gen

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Introduction

This guide describes the CA Gen .NET Server generation feature. This feature allows you to generate, build, and assemble your Server Procedure Steps as serviced components that execute as a COM+ application under the control of Component Services.

This chapter provides an overview of Microsoft .NET technology and how it is used by CA Gen. Later chapters provide task-related information and reference materials.

Microsoft .NET

Microsoft .NET is an application development, deployment, and runtime platform for a wide variety of applications. Microsoft .NET consists of the .NET Framework, development tools, client, and server systems. The .NET Framework is used to develop and execute applications. Tools such as Microsoft Visual Studio provide a sophisticated IDE to allow developers to build applications quickly. The client systems provide the distributed user interface while the server systems provide a scalable, back-office computing resource.

Terminology

Microsoft .NET introduces a number of new products, concepts, and terms that are defined in the following sections.

.NET Framework

The .NET Framework is a set of software from Microsoft to build and run applications on Windows systems. The .NET Framework includes the C# compiler, the CLR, and many other tools.

Microsoft Visual Studio

Microsoft Visual Studio is a sophisticated IDE that can be used in conjunction with the .NET Framework to visually develop, debug, and package .NET applications.

C#

C# (pronounced C Sharp) is a new programming language created by Microsoft in the year 2000. It is a high level, strongly typed, object oriented language with similarities to C++ and Java. A C# source file has the .cs file extension, for example, hello.cs.

C# allows a namespace to be defined so that classes from different origins do not conflict. For example, CA could produce a class named Address. By declaring a namespace name when the class is defined (com.ca.gen), the effective class name becomes com.ca.gen.Address. This effective name would be unique even in the case of another company creating their own class named Address since both companies should have unique namespace names. The System namespace is a special namespace reserved to Microsoft.

In C# and other .NET-aware compilers, all strings are encoded in Unicode.

Rather than generating machine-specific object code, the output from the C# and other .NET-aware compilers is Microsoft intermediate language (MSIL). MSIL is a CPU-independent "assembly code" that will be converted to native machine code at runtime.

The C# and other .NET-aware compilers generally produce ready-for-execution assemblies rather than object modules that require link editing. An assembly may either be a DLL or an EXE. Assemblies may also contain resources, security, and versioning information.

In addition to assemblies, an application may have one or more configuration files. Configuration files are XML files that contain additional information about the application and the environment it executes within. Configuration files may specify information such as the private path or security settings and can be changed without recompiling the application.

The source C# or other .NET-aware code can be written to the Common Language Specification (CLS). This specification allows assemblies to be callable by any .NET-aware language. For example, following the specification allows a VB.NET application to invoke an assembly written in C#.

Common Language Runtime

The Common Language Runtime (CLR) provides the runtime environment for compiled MSIL code. The CLR provides services including just-in-time compilation into native machine code, memory management including garbage collection, thread management, and security services.

Code that executes within the CLR is called managed code. Managed code relies on the CLR for its execution environment. Unmanaged code generally executes on the native hardware and performs its own memory management and other services. Microsoft has provided facilities to allow managed and unmanaged code to interoperate with each other.

Strong Naming

Microsoft has introduced strong naming to improve security and prevent DLL conflicts. When a DLL is strongly named, it contains several attributes that uniquely identify it such as version number and an encryption key. The caller of a strongly named DLL must also be strongly named thus extending the security throughout the application. This ensures that the caller of a method within a DLL (assembly) will execute the one it was assembled against.

Local and Global Assembly Cache

All assemblies and other resources required by the .EXE are generally required to be in the local assembly cache or the global assembly cache. It is important to note that the PATH variable is not used.

When a .NET application begins execution, the CLR begins the process of probing (loading) the DDLs required by the application. The order of probing is (1) the GAC (2) the directory where the .EXE is located and (3) the directory or directories specified by the privatePath attribute in the application configuration file.

Local Assembly Cache

The local assembly cache is the application root directory plus the subdirectories that are specified in the privatePath attribute.

The application root directory is the directory where the .EXE file is located. If the application has configuration files, they must be located in this directory.

One or more subdirectories under the application root directory may contain additional assemblies or resources required by the application. Specifying the directory or list of directories in the privatePath attribute causes them to be searched during probing.

Global Assembly Cache

The global assembly cache (GAC) is a single location on the system where you can share assemblies. The GAC stores assemblies in a set of directories located under the <windows-root>\assembly directory, but this set of directories appears as a single directory in Windows Explorer.

The GAC can only contain assemblies. Before you can store an assembly in the GAC, the assembly must be public and strongly named. If an assembly is present in the local assembly cache and the GAC, the copy stored in the GAC is used.

Note: You cannot store common data or configuration files in the GAC.

Assemblies loaded from network shares are treated differently than assemblies loaded from local drives. The assemblies loaded from network shares are considered less trusted. Security settings may require changing in order for applications to be executed from a network share.

ADO.NET and Data Provider Classes

ADO.NET is a set of classes that allow applications to access database management systems such as Microsoft SQL Server, Oracle, and DB2.

Note: ADO.NET is not a DBMS; it is only an access mechanism.

Data providers are a set of classes that act as a bridge between ADO.NET and the DBMS. Data providers are DBMS-specific and may be provided by Microsoft, the DBMS vendor or a third-party.

Internet Information Server

Internet Information Server (IIS) is the HTTP and FTP server from Microsoft for Windows platforms. For .NET servers using .NET Remoting, IIS is used to receive requests from the network and pass them on to COM+ for execution by the target server application.

Component Services (COM+)

COM+, also known as Component Services, is the next-generation Microsoft Component Object Model (COM) and Microsoft Transaction Server (MTS). COM+ provides services and resource management to applications including role-based security, object pooling, just-in-time activation, transaction support, and coordination.

Serviced Components are classes that are created by C# or other .NET-aware compilers and use COM+ services. A set of serviced components is referred to as a COM+ application.

.NET Remoting

.NET Remoting is a .NET-native remote procedure call from a .NET client (such as ASP.NET Web Forms pages) to a .NET server (such as a serviced component executing under COM+). .NET Remoting is designed to easily pass through the firewall by using HTTP on port 80.

MSI File

An MSI file contains an application ready for installation on a Windows system. The Microsoft Installer processes the .MSI file and performs all necessary file operations including the tracking of shared components (that is, DLLs). All applications installed with Microsoft Installer may be removed from the system using the Add or Remove Programs option on the Control Panel.

CA Gen Applications—C and COBOL Generation

Over the years, CA Gen has evolved from an application development tool that generates block mode and batch applications to one that generates client/server applications. In the latter case, the application is divided into the following elements:

- GUI—The application's graphical user interface.
- Client Procedure Steps—Responsible for handling the GUI events and performing some of the application logic. They are usually related to the flow of the application and may access the application's database. Utilizing middleware, Client Procedure Steps invoke Server Procedure Steps to perform the balance of the application functionality.
- Server Procedure Steps—Responsible for performing the bulk of the application's business logic and database accesses. Server Procedure Steps do not have a user interface. If allowed by the TP monitor, a Server Procedure Step can invoke other Server Procedure Steps.

For Windows-based GUI Client/Server applications, the GUI and Client Procedure Steps are generated in the C language for execution on a Microsoft Windows platform. The Server Procedure Steps are generated in either COBOL for execution on the IBM mainframe or in the C language for execution on Microsoft Windows or selected Linux or UNIX platforms.

The GUI Clients can communicate with the Servers using one of many communications runtimes, including:

- The CA Gen Client Manager
- TCP/IP
- Oracle Tuxedo (Tuxedo)
- WebSphere MQ
- ECI

Note: Not all servers support all the above communications methods. For more information on cooperative flows, see the *Distributed Processing – Overview* guide.

C# Generation

The CA Gen code generators have been enhanced to generate C# code. CA Gen can generate a complete CLS compliant .NET application from a CA Gen Client/Server model. Note that prior to generation, you should add .NET-specific information to your models such as the namespace names and, when generating servers, the attributes of strongly named assemblies including the assembly key pair. This guide describes these tasks.

An action diagram is generated as a class whose name is the CA Gen member name. Further, all import, export and local views are generated as separate classes. For example, the action block DEMO_AB will be generated as the class DEMOAB. Its import, export and local views are generated as the classes DEMOAB_IA, DEMOAB_OA and DEMOAB_LA, respectively. An additional set of C# files are generated by the Window Manager and Server Manager. These classes present the import and export views in a more object-oriented manner. Additionally, wrapper classes are created to move the data into and out of the _IA and _OA classes.

The generated C# code uses ADO.NET to access the application database. ADO.NET provides a consistent API to data providers which are DBMS-specific. CA Gen supports selected data providers for popular DBMSs such as IBM DB2, Microsoft SQL Server and Oracle. Additionally, the ODBC data provider is supported to gain access to additional DBMSs.

C# Runtime

The generated C# code is supported by a CA Gen C# runtime. The architecture of the runtime is similar to that of the Java runtime but written entirely C#. To improve performance, the runtime caches objects for reuse whenever possible. In the .NET Server environment, the runtime uses the facilities provided by the COM+ services to cache objects such as resource management, transaction control, and flow control.

Supported Features

The generated C# is designed to be functionally equivalent to generated C, COBOL, and Java code. Accordingly, the generated C# supports:

- CA Gen Referential Integrity
- DBMS-enforced Referential Integrity
- For appropriately marked attributes, decimal precision to 18 digits by using the .NET decimal class
- All 6-digits in the microsecond field of a CA Gen timestamp
- Debugging with the CA Gen Diagram Trace Utility
- Runtime logging with CMI Debug
- All relevant user exits
- Generation of external action blocks

Multiple Database Support

CA Gen-generated C applications can access only one database for a given procedure step. This is due to the use of the default database handle for all embedded SQL statements.

CA Gen-generated C# applications use ADO.NET and have been designed to allow access to one or more databases within a given procedure step. Customers wishing to use this feature must create a data source for each database to be accessed by the procedure step.

Note: The commits and rollbacks of the multiple database connections may or may not be performed using a 2-phased commit facility. In the ASP.NET Web Client environment, 2-phased commit is not supported by ASP.NET or IIS. Therefore, be aware that there is a small risk of data integrity problems should an error be encountered. For the generated .NET Server environment, 2-phased commit is supported when utilizing properly configured data sources.

ASP.NET Web Client Generation

CA Gen can generate Client Procedure Steps for the ASP.NET Web Forms environment. The application's GUI is generated as XML utilizing server side controls and the Client Procedure Steps are generated as C# and ASP.NET Web Forms pages. ADO.NET is used for all database accesses in the generated C# code.

The Client Procedure Steps generated as ASP.NET Web Clients are allowed to invoke Server Procedure Steps generated as C, COBOL or C# code.

The Client Procedure Steps may interact with C or COBOL-generated Server Procedure Steps using the TCP/IP communications runtime.

CA Gen .NET Server Generation

CA Gen can generate the Server Procedure Steps as serviced components executing under COM+. For simplicity, these are referred to as CA Gen .NET Servers.

As with the C# generated for the Client Procedure Steps, ADO.NET is used to access all databases.

Client-to-Server Flows

ASP.NET Web Clients and .NET Proxy Clients

The CA Gen ASP.NET Web Clients and CA Gen .NET Proxy Clients may communicate with the CA Gen .NET Servers using the CA Gen .NET Remoting Communications Runtime. Rather than using the CA Gen Common Format Buffer, the CA Gen .NET Remoting Communications Runtime uses the data interchange and remote invocation mechanisms native to the .NET Framework. Specifically, the runtime serializes the calling client's views and invokes the target CA Gen .NET server using a.NET Remoting call. In client-to-server flows, the client does not pass its transaction context to the server. Therefore, a CA Gen-generated .NET Server will start its own transaction context.

MFC GUI Clients, C, and COM Proxy Clients

The CA Gen C-language GUI Clients as well as the CA Gen C and COM Proxy Clients must use the CA Gen C to C# Cooperative Flow to communicate with the CA Gen .NET Servers. This cooperative runtime performs the conversion and .NET Remoting call on the client system. Accordingly, the client system must have the .NET Framework installed and a local copy the target generated CA Gen .NET Server's interface assemblies.

More information:

Cooperative Flows (see page 47)

Hand-Written .NET Clients

It is possible for hand-written .NET applications to access generated CA Gen .NET Servers without utilizing a CA Gen-generated Proxy. The interface of the CA Gen-generated .NET Server performs all required data validations prior to executing the target procedure step.

Server-to-Server Flows

In server-to-server flows, a CA Gen-generated .NET Server can invoke another CA Gen-generated .NET Server using the DCOM Communications Runtime. This runtime is used specifically to support server-to-server cooperative flows as .NET Remoting does not support distributed transactions. For simplicity, the CA Gen Workstation Toolset does not call out the DCOM Communications Runtime separately. Rather, the CA Gen displays only .NET Remoting but use DCOM when appropriate.

Note: CA Gen DCOM Cooperative Flow cannot be used on client-to-server flows. Also note that server-to-server flows may not pass easily through the firewall.

The calling CA Gen .NET Server can be in the same machine as the target CA Gen .NET Server, or it can be located elsewhere on the network (as supported by the .NET Framework). The target CA Gen .NET Server can start its own transaction context, or it can extend the transaction context of the calling CA Gen .NET Server as specified in the CA Gen model.

CA Gen-generated .NET Servers can invoke a CA Gen C or CA Gen COBOL servers using the CA Gen TCP/IP, Web Services, or MQSeries communications runtime.

Distributed transactions are not supported when servers are executing within different environments (TP systems).

Calling non-CA Gen-generated .NET Servers (services components) can be accomplished through External Action Blocks.

The 32K Limit

The 32K Limit has been designed out of the C# generated code and supporting runtimes. There are no design limits for flows between ASP.NET Web Clients and CA Gen .NET Servers when using the CA Gen .NET Remoting Communications Runtimes. Likewise, there are no design limits for server-to-server flows between generated .NET Servers using the CA Gen DCOM Communications Runtimes.

Restrictions When Targeting Generated .NET Servers

All asynchronous Action Diagram statements are not supported and will cause a generation error.

Supported Windows Platforms

.NET Applications created by CA Gen are designed to be CLS-compliant.

Note: For a complete description of the Windows platforms supported by CA Gen, see the *Technical Requirements* document.

Generating a Model in C#

The following sections describe the process of generating and assembling a CA Gen client/server model.

Generation

The generation of CA Gen ASP.NET Web Clients and CA Gen .NET Servers may take place on the CA Gen Workstation Toolset or the CA Gen Client/Server Encyclopedia (CSE). If the generation takes place on a CSE, the remote files must be moved to a supported Windows workstation to be built.

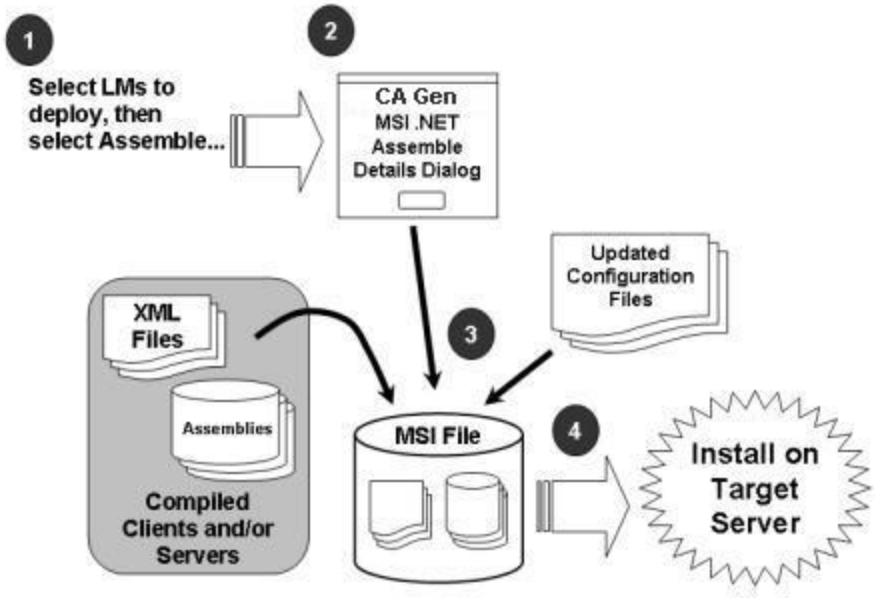
Build

Once the application has been generated in C#, it can only be built on a supported Windows system using the CA Gen Build Tool.

After the cascade and all load modules have been successfully built, the application is ready for assembly into an MSI file.

Assembly

During assembly, the files that make up a CA Gen .NET Application are placed into an MSI file. Later, the MSI file is invoked and installed using standard Windows tools. The following diagram shows how this is accomplished.



Once the Cascade Library and Load Modules to be assembled into an MSI file have been selected, select Action, Assemble or click the Assemble toolbar icon to open the MSI .NET Assemble Details dialog.

Note: You must not select the DDL module.

1. The MSI .NET Assemble Details dialog allows you to specify the properties of the MSI file including its name.

   If an ASP.NET Web Client has been selected for assembly, you can specify the client's properties.

   For a detailed explanation of this dialog, see MSI Assembling in the *Build Tool* Guide.

2. The Build Tool executes a script that performs the following tasks:

   - The files of the selected CA Gen ASP.NET Web Clients and/or CA Gen .NET Servers are placed into the MSI file.
   - The generated configuration files are updated with information from the MSI .NET Assemble Details dialog.

   Upon completion, the CA Gen–generated .NET Application is ready for installation on the target system. If necessary, copy the MSI file to the target system.

3. Activating the MSI file will cause the installation to be performed. The installation automatically creates the IIS virtual directory needed to invoke the CA Gen .NET Servers as well as register the CA Gen .NET Servers with COM+. After the installation has completed, the application is ready for execution.

Testing Recommendations

Applications with CA Gen .NET Servers should be installed on a small scale system for local testing. If Diagram Trace is to be used, it is recommended that it take place in a single user environment.

COBOL Application Considerations

This section addresses re-targeting COBOL-generated applications to either C, C#, or Java. COBOL has the capability of storing and manipulating numbers with precision. The native C, C# or Java data types have limitations in either the number of digits or their precision. For example, most numbers with decimal digits will be generated as a double, which has a limit of 15 digits of precision and can lose precision when used in some mathematical operations.

CA Gen has the capability of generating an alternative data type that can achieve 18 digits of precision. In C, CA Gen uses a character array and a special software library. In C#, CA Gen uses the decimal type from the system namespace. In Java, CA Gen uses the BigDecimal class from the java.math package.

Using these alternative data types consumes more CPU resources, as it uses a software library to perform mathematical operations, whereas, the native double data type uses hardware instructions to perform the same mathematical operations.

CA Gen does not generate these alternative data types automatically. Rather, you should examine your data model and determine which attributes should be generated in this alternative form and set the appropriate option.

After an attribute has been identified, its properties dialog should be displayed by either double-clicking on the attribute or selecting the attribute and then choosing Properties from the Detail menu. To instruct CA Gen to generate the alternative data type, select the Implement in C, C#, and Java with decimal precision check box.

DBMS Considerations with Decimal Precision

For C# generated code, CA Gen implements all attributes marked as decimal precision as an instance of the decimal class and passes them to the ADO.NET data provider. It is the responsibility of the data provider to accurately store and retrieve the application data.

Unicode Considerations

Unicode is a method for encoding characters that allows one application to process characters and strings from most of the languages of the world. Nearly one million unique characters can be represented in Unicode. Unicode is neither a single byte character system (SBCS) nor a double byte character system (DBCS).

The C# and Java languages support only Unicode for characters, literals, and strings represented in code. This is true for all C# and Java applications, not just those generated by CA Gen.

It is possible for users of C# or Java-generated applications to provide input data that cannot be stored on a database that is not configured to store Unicode.

Customers should evaluate the advantages and disadvantages of converting the applications database to Unicode. If the database is not converted, C# and Java applications could submit unrecognized character that cannot be stored correctly. Alternatively, customers may choose to use only C and COBOL applications with a database that is set to store DBCS characters.

This creates challenges when you attempt to use C# and Java code to access databases. Most existing databases store characters in a codepage that represents a subset of Unicode characters. Thus many of the characters processed by the C# application cannot be represented on the database. Unsupported characters are usually replaced with a 'sub character' or a '?' depending upon the database translation algorithm.

Converting existing databases to Unicode is possible with most implementations and would allow all characters entered by a user and processed by the code to be stored. However, converting a database to use Unicode could require changes to existing applications and in the way they access the database.

For a CA Gen application that creates a new database, CA suggests that the new database always support Unicode encoding. Two major Unicode encoding schemes are supported by most databases:
UTF-8 and UCS-16. UCS-16 Unicode Coding Sequence 16-bit is the simplest because every character in the database is represented as an unsigned 16-bit number, which is the same representation as characters in a C# and Java application. UCS-16 uses greater amounts of storage when most of the characters represented in a database are indicated by code point numbers less than 128 (0x80). UTF-8 (Unicode Transfer Format 8-bit) solves this problem because all code points less than 128 require only a single 8-bit byte of storage (code points between 128 and 2047 require two bytes, and code points between 2048 and 65535 require 3 bytes and so on).

Further, a database created to support UTF-8 makes more assumptions about the type and size of characters in each text column. For example, a text column that is fixed ten characters (bytes) in length correctly stores ten characters when all ten code points are less than 128. But any higher character code points would require more than ten bytes in the column. When the type of data is unknown, all character columns in a UTF-8 database should be defined as variable and three times the length of the maximum string written by the C# application (in this example, 3*10 or thirty).

Chapter 2: Software Installation and Configuration

This chapter describes the tasks that must be performed before, during, and after the installation of CA Gen, with the intent of performing generation of Server Procedure Steps as .NET Servers.

Installing Prerequisite Software

This section describes the third-party software products used by the CA Gen .NET Servers.

Construction Workstation

The software products in the following list must be present on the Windows system being used to build, assemble, and execute the CA Gen-generated .NET Servers.

Note: For the supported versions of software, see the *Technical Requirements* document.

- Windows Operating System
- Internet Information Services (IIS)
- Microsoft .NET Framework
- Microsoft Visual Studio .NET (required to assemble the generated application into an MSI File)
- ADO.NET Data Provider (if not provided by the Microsoft .NET Framework)

Client/Server Encyclopedia

The Client-Server Encyclopedia has no special requirements for CA Gen .NET Server generation.

Runtime Environment

The following are the software products that must be present on the Windows system being used to install and execute the CA Gen-generated .NET Servers.

Note: For the supported versions of software, see the *Technical Requirements* document.

- Windows Operating System
- Internet Information Services (IIS)
- Microsoft .NET Framework
- ADO.NET Data Provider (if not provided by the Microsoft .NET Framework)

CA Gen Installation

The following sections describe the tasks to install CA Gen with support for generating, assembling, and deploying applications with .NET Server.

Windows Systems

The following sections describe the tasks to install CA Gen with the intent to generate, assemble and deploy .NET Servers on a Windows system.

Pre-Installation Procedures

The following tasks should be performed before installing CA Gen:

Follow these steps:

1. Obtain the CA License file for your system.

   Note: For more information, see the *Distributed Systems Installation Guide*.

2. Install all prerequisite software.

3. To execute properly, the CA Gen Workstation Toolset, build scripts and batch utility files require the .NET Framework and Visual Studio tools to be accessible from the PATH. The environment variable VS##COMNTOOLS is used to make the Visual Studio tools accessible from a DOS prompt.

   To ensure your system is correctly configured, open a DOS Prompt. Enter the command SET VS. If successful, you should see an environment variable definition that references the Visual Studio tools subdirectory.

   If the preceding command fails, Visual Studio is not installed or not configured correctly. Troubleshoot the problem and consider reinstalling Visual Studio.

Tools to Install

The CA Gen typical installation installs all the software necessary for the Windows workstation to generate and assemble a CA Gen .NET Application.

Post-Installation Procedures

There are no post-installation procedures.

Client/Server Encyclopedia (Windows or UNIX)

The following sections describe the tasks to install the Client/Server Encyclopedia with the intent to generate, assemble and deploy .NET Servers on a Windows or UNIX system.

Pre-Installation Procedures

Before you install CA Gen, be sure to obtain the CA License file for your system.

Note: For more information, see the *Distributed Systems Installation Guide*.

Installation Procedures

During the CA Gen installation, a list of available CA Gen components is displayed. The Encyclopedia Server and Encyclopedia Construction Server should be installed to generate CA Gen .NET Servers from the Client/Server Encyclopedia.

Post-Installation Procedures

There are no post-installation procedures.

Installing the Data Provider

A data provider is a class or set of classes that provide a bridge between ADO.NET and the desired data source (such as a DBMS). Data providers can be supplied by Microsoft, DBMS vendors, or third-parties.

Unless you plan to use a data provider supplied by Microsoft, you will need to install the data provider.

Note: CA Gen does not support all data providers. For a list of supported data providers and their versions see the *Technical Requirements* document.

Chapter 3: DDL Generation

CA Gen-generated .NET applications do not require a .NET-specific DDL Generation. All generated .NET applications use ADO.NET to access their database and have no special requirements on how the DDL is generated or installed.

In general, customers who are re-targeting their application to .NET should continue to use the Technical Design of their existing application. For example, if the existing application is generated in C and uses Oracle, then the CA Gen .NET Servers should be generated with Oracle as the DBMS setting. Consequently, existing CA Gen-generated databases can be reused by CA Gen-generated .NET applications.

Using the ODBC/ADO.NET Technical Design

The ODBC/ADO.NET Technical Design can be used when there is no existing technical design to reuse.

The following steps are required to install DDL using the ODBC/ADO.NET Technical Design:

Follow these steps:

1. Set up the DBMS.
2. Define the ODBC Data Source.
3. In the CA Gen Build Tool, define the User ID, Password, and other information needed to access the database.
4. Perform DDL Generation with Installation.

Set Up the DBMS

Perform the following steps to set the DBMS.

Follow these steps:

1. Install and configure the DBMS software.
2. Perform all post-installation tests to ensure the DBMS is ready for use.
3. Create the physical database and table spaces required to host the tables and indexes.

If necessary, set the security within the DBMS so that the CA Gen user or application can access the database.

Define the ODBC Data Source

Using the Windows tools, define the ODBC Data Source to be used by your application.

Set Up the Build Tool

Start the CA Gen Build Tool to define the following token keys:

Follow these steps:

1. Open the profile in the Profile Manager.
2. Expand DBMS.
3. Select ODBC.
4. Set OPT.DBCONNECT to the ODBC connection string for the target database
5. Set OPT.DBUSER and OPT.DBPSWD to access the target database.

Performing DDL Generation

Use the following procedure to perform the DDL generation and installation:

Follow these steps:

1. Start the CA Gen Workstation Toolset
2. Open the model.
3. Open the Technical Design Diagram.
4. Select and open the ODBC/ADO.NET Technical Design.
5. If necessary, perform transformation or retransformation to synchronize the logical and physical data models.
6. Open the Generation Diagram.
7. Open the Generation Defaults dialog. Set the generation parameters:
   - Operating System: Windows or CLR
   - DBMS:ODBC/ADO.NET
8. Perform the DDL Generation and Installation.

Chapter 4: Pre-Code Generation Tasks

This section contains the following topics:

Model Considerations

This section details the following areas of concern when generating the Server Procedure Steps as .NET Servers:

- Marking attributes to use Decimal Precision
- Adding .NET-specific information to the CA Gen model
- Setting .NET Remoting-specific properties for Server Managers

Decimal Precision

If you are re-targeting your COBOL application to C#, you may want to set some numeric attributes to use Decimal Precision.

More information:

Introduction (see page 9)

Adding .NET-Specific Information to a Model

There are four .NET-specific pieces of information that can be added to a model:

- Namespace name (required when using CBD techniques)
- Application name
- Assembly version number
- Assembly key pair (required for servers)

The following sections describe each .NET-specific piece of information in detail.

Namespace Names

C# allows a namespace to be defined so that classes from different origins will not conflict. For example, CA could produce a class named Address. By declaring a namespace name when the class is defined (com.ca.abc), the effective class name becomes com.ca.abc.Address. This effective name would be unique even in the case of another company creating their own class named Address since both companies should have unique namespace names.

Although most situations do not require you to set the .NET namespace name, you may find it useful to do so.

Note: Customers who use CBD techniques must set the .NET Namespace Names in their models.

By convention, the namespace name is the reverse of the company's Internet domain name. For example, www.ca.com would become com.ca.<facility-name>.

Namespace names can be specified at the model and business system levels. If no namespace name is specified at the business system level, the namespace name at the model level is used. If no namespace name is specified at either level, a name is derived. If the generation is taking place on the Workstation, a version of the short model name is used. If the generation is taking place on the CSE, the name used is m<model-id>.

Namespace Names for CBD Models

To allow components to be easily consumed in the C# environment, the following namespace name settings are recommended:

- In the implementation model, define a Namespace Name at the Model Level.
- In the consuming model, import the spec model into its own Business System and set its namespace name to that of the implementation model.

Application Name

.NET applications may be given a name. The name is used in the URL when invoking the application.

An application name can be specified only at the model level. If no application name is specified at either level, a version of the short model name is used.

Assembly Version

.NET assemblies may contain an assembly version as part of its strong name.

An assembly version can be defined at the model or business system levels. If no assembly version is defined at the business system level, the Assembly Version at the model level is used. If no assembly version is defined at either level, the value 0.0.0.0 is used.

The assembly version is actually a string consisting of four numbers separated by periods. For example:

1.2.345.0

Microsoft has defined the four parts of the assembly version as follows:

- The first number (1 in the example above) is the major version.
- The second number (2 in the example above) is the minor version.
- The third number (345 above) is the build number.
- The fourth and last number (0 above) is the revision number.

Note: The assembly version must match exactly in order for the runtime to consider it a match. Therefore, changing the revision number would cause the assembly versions to mismatch and cause the assembly to not load at runtime.

Assembly Key Pair

.NET assemblies may be signed with an assembly key pair. Since the servers created by CA Gen are strongly named, an assembly key pair must be defined.

An assembly key pair can be defined at the model or business system levels. If no assembly key pair is defined at the business system level, the assembly key pair at the model level is used. If no assembly key pair is defined at either level, the Server Manager generator stops execution with an error message.

The assembly key pair is actually a string of about 1024 bytes in length. CA Gen can generate these keys for you. On the dialogs described below, click Create. Further, the key pairs may be exported to or imported from other models by clicking the appropriate buttons.

Setting the Model's .NET-Specific Information

To set the .NET-specific information at the model level,

Follow these steps:

1. In the Workstation Toolset with the model open, open the Environment Diagram by using one of the following methods:
   - On the main menu bar, select Construction, then Environment.
   - On the main menu bar, select Tool, then Construction, then Environment.
   - On the tree control, select the Diagram tab, expand Construction, and then double-click Environment.
2. On the Main Menu bar, click Options, then Model Generation Properties.
3. Enter the .NET-specific information in the fields provided and then click OK.

Setting the Business System-Specific Information

To set the .NET-specific information at the business system level, use the following procedure:

Follow these steps:

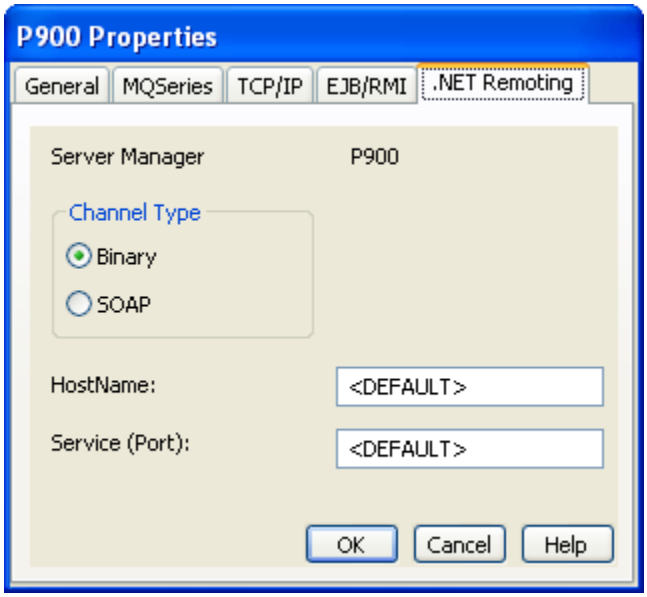
1. In the Workstation Toolset with the model open, open the Environment Diagram by using one of the following methods:
   - On the main menu bar, select Construction, then Environment.
   - On the main menu bar, select Tool, then Construction, then Environment.
   - On the tree control, select the Diagram tab, expand Construction, and then double-click Environment.
2. Select the target business system.
3. On the main menu bar, click Detail, then Properties.
4. Enter the .NET-specific information in the fields provided and then click OK.

Server Manager Properties

In order for generated clients to find a CA Gen .NET Server, the clients need to know the host and port number of IIS where the servers are executing. Clients can retrieve this information from either the Server Manager Properties or from the COMMCFG files.

The Server Manager Properties dialog allows you to specify this .NET Remoting-specific information. The values that are entered in the fields tend to be server dependent. Therefore, the MSI file created by CA Gen will contain server-specific information.

To open the Server Manager Properties dialog, select the Cooperative Packaging or the Cooperative Generation dialog. Then either double-click the desired Server Manager, or select the Server Manager and select Properties from the Detail menu. The Server Manager Properties dialog opens.



Enter the appropriate values as follows:

- Channel Type—Defines the format of the data to be sent via .NET Remoting. In general, Binary should be selected.
- HostName—The name or IP address of the remote system hosting the CA Gen-generated .NET Server
- Service (Port)—The port number of IIS of the remote system hosting the CA Gen-generated .NET Server. Typically, the value is 80.

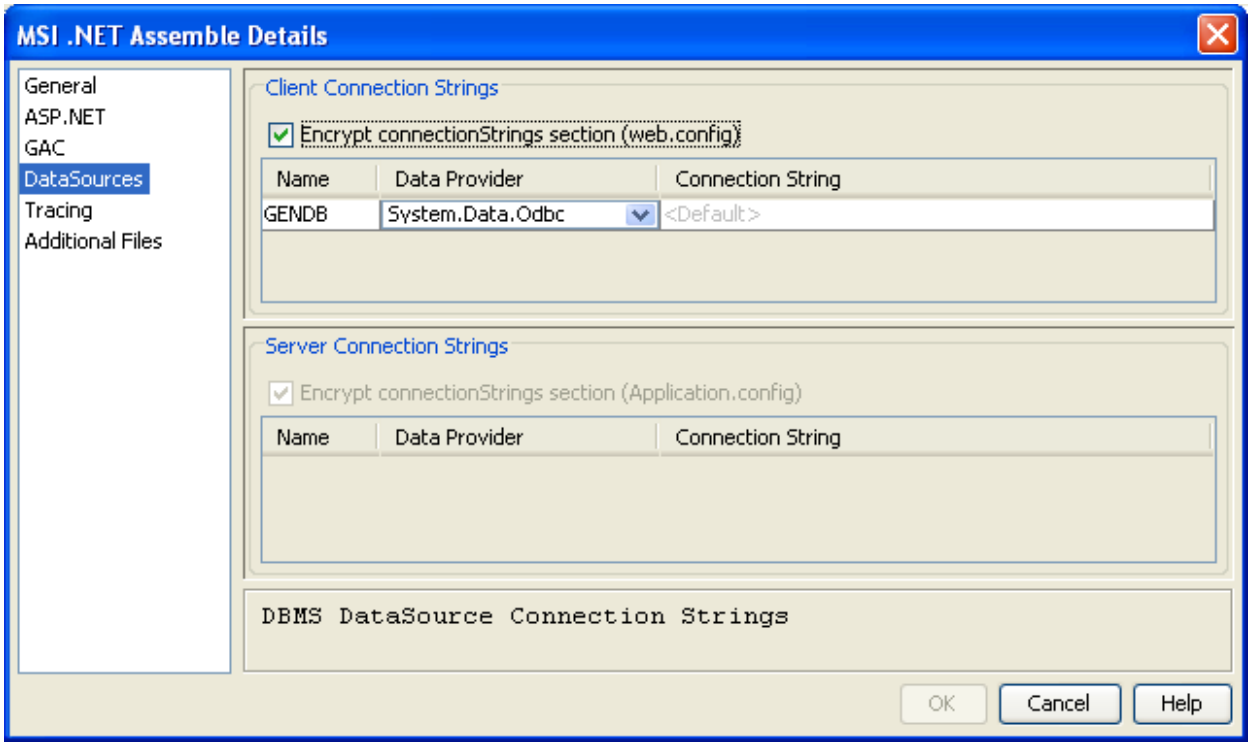
Configuring the Build Tool

CA Gen .NET Servers do not require any settings to be made in the CA Gen Build Tool.

Configuring the Data Source

A data source is the target of a data provider. You can configure the data source while assembling the MSI file in the Build Tool.

When you select DataSources from the MSI .NET Assemble Details panel, the following dialog appears:



The fields in the Tracing dialog are as follows:

Client Connection Strings

Specifies the connection string parameters for every logical database name specified with ASP.NET Clients.

Encrypt connectionStrings section (web.config)

If selected, CA Gen encrypts the connection string in the web.config file

Name

Specifies the name of the database

Data Provider

Specifies the data provider. This is an enterable drop down that displays the list of data providers. By default, four data providers are listed. They are:

- System.Data.Odbc
- System.Data.SqlClient
- IBM.Data.DB2
- Oracle.DataAccess.Client

Note: You can enter your own data provider

Server Connection Strings

Specifies the connection string parameters for every logical database name specified with .NET Servers.

Encrypt connectionStrings section (Application.config)

If selected, CA Gen encrypts the connection string in the Application.config file

Name

Specifies the name of the database

Data Provider

Specifies the data provider. This is an enterable drop down that displays the list of data providers. By default, four data providers are listed. They are:

- System.Data.Odbc
- System.Data.SqlClient
- IBM.Data.DB2
- Oracle.DataAccess.Client

Note: You can enter your own data provider.

Configuring the COMMCFG.TXT File

CA Gen-generated ASP.NET and .NET Proxy clients can use the COMMCFG.TXT file to define the communication infrastructure and location of the target servers.

If you did not specify Server Manager Properties for all servers in your application, you must configure the COMMCFG.TXT file.

Using the editor of your choice, open the file COMMCFG.TXT in the .net subdirectory under the directory where CA Gen is installed:

```
cd %GENxx%Gen
cd .net
write commcfg.txt
```

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Review the comments in the file for the available cooperative flows and their parameters. Enter records in the file to allow the generated client to access the target server.

For example, the following will cause all transactions to be sent to the CA Gen .NET server using the CA Gen .NET Remoting Communication Runtime to localhost on port 80:

```
* = net localhost 80 b
```

Note: For a detailed discussion on the use of the COMMCFG.TXT file, see the *Distributed Processing – Overview Guide*.

Chapter 5: Construction, Assembly and Installation

This chapter describes the tasks for generating, assembling and installing a CA Gen application for the .NET environment.

Prerequisites

The following tasks should be completed prior to generating your application for the .NET environment:

- The application database must be installed and ready for use.
- The model must have been modified to define an assembly key pair. Additional modifications to the model may be performed as described in the prior chapter.
- The data provider and data source for the application database must already be defined.

Generation

This section describes the actions required to generate CA Gen Server Procedure Steps as .NET Servers.

The generation can take place on the Windows workstation toolset or the Client Server Encyclopedia. The construction of the generated code can only be performed with the Build Tool executing on a properly configured Windows platform. Therefore, the remote files created by the CSE must be copied to a system with the CA Gen Build Tool to construct the application.

On the CA Gen Windows workstation, all code is generated into the <model-directory>\c# directory.

Target Environment Definition

When generating CA Gen Server Procedure Steps as .NET Servers, set the following generation parameters in the Server Environment Parameters dialog:

Parameter	Value
Operating System	CLR

Parameter	Value
DBMS	ODBC/ADO.NET or any supported DBMS (Technical Design)
Language	C#
TP Monitor	Comp_Services
Communications	.NET Remoting

Setting the DBMS Generation Parameter

When setting the DBMS Generation parameter, ODBC/ADO.NET or any valid DBMS can be chosen. Regardless of the setting, all database accesses are generated as ADO.NET calls.

The only advantage to choosing a specific DBMS over ODBC/ADO.NET is that choosing a specific DBMS uses the table names, index names, and other user-modified names that have been defined under the Technical Design dialogs. Therefore, it may be better to choose a specific DBMS when the application will access an existing database. Applications accessing a new database may want to choose ODBC/ADO.NET.

Construction

Building CA Gen .NET Servers is very similar to building any other generated application. However, the construction of the generated application can only be performed with the CA Gen Build Tool running on a Windows system.

If the application was generated on the Windows workstation, the CA Gen Build Tool is invoked to build all generated cascade and load modules.

If the application was generated by a CSE, copy the remote files to a Windows system with the CA Gen Build Tool. Then start the Build Tool, select the directory where the remote files are located, and click Add Module. This will cause all remote files to be loaded into the CA Gen Build Tool's module panel. Next, select all cascade and load modules to be built and then click Build.

The results of the construction process are placed in the <model-directory>\c#\build directory. The following files can be found in the build directory:

File	Description
<LM-Name>.dll	The assembly containing the compiled C# code.
<LM-Name>. components.dll	Created for CA Gen .NET Servers only, this file contains the generated Server Manager code.

<LM-Name>.interfaces.dll	The portable interfaces for the public methods in <LM-Name>.dll.
<LM-Name>.resources.dll	Created for CA Gen ASP.NET Clients only, this file contains the XML used to describe the web pages.
<LM-Name>.web.config	The configuration file for the load module.

Assemble

After all code has been successfully built, the application is ready to be assembled into an MSI that will be used to install the application on the target server. The assemble process can only be performed by the CA Gen Build Tool running on a Windows system. Procedures for assembling and installing the generated application follow.

Selecting Load Modules for Deployment

Follow these steps to assemble the cascade library and load modules into an MSI file:

Follow these steps:

1. In the CA Gen Build Tool, select the Cascade Library, ASP.NET Web Clients and .NET Server load modules to be assembled from the model. To select all load modules, click the model's title line.

   Note: Do not select the application's database.

2. Once all the desired libraries and load modules have been selected, select the menu Action, Assemble or click the Assemble toolbar icon. The MSI .NET Assemble Details dialog opens.

   Note: For information about how to complete this dialog, see the *Build Tool User Guide*.

3. After the assemble dialog has been completed, click OK to have the CA Gen Build Tool create the MSI file.

Review the Assemble Status

After the CA Gen Build Tool has completed the assemble process, the log file should be reviewed. Select the assemble entry in the Build Tool (generally the last line in the module panel), and click the Review toolbar icon to display the contents of the log file.

Review the assemble log file to ensure the process completed successfully.

The log file is located in the following directory:

```
<model-directory>\c#\assemble.MSI.NET.out
```

Saving the Assemble Process Information

The CA Gen Build Tool creates the file assemble.MSI.NET.icm in the <model-directory>\c# directory to preserve the information entered on the MSI .NET Assemble dialog. Customers may want to preserve and manage this file since it contains the assemble process-specific information.

Installing the Generated Application

After the CA Gen Build Tool has created the MSI file, the .NET application is ready for installation. The MSI file is located in the <model-directory>\c#\deploy directory. Double-click the MSI file to perform the installation.

If necessary, copy the MSI file to the target system. If a prior copy of the application with the same GUIDs is present, it must be uninstalled before reinstalling the new MSI file. Uninstall the old version of the application with Start, Settings, Control Panel, Add and Remove Programs.

Invoke the MSI file. One way to accomplish this task is to locate the MSI file with Windows Explorer and double-click it. An MSI Installation Wizard appears to lead you through the remaining installation steps. Executing the windows installer package installs your application under IIS.

After the installation has completed, the application is ready for use.

Troubleshooting Installation Errors

Should the MSI installation fail, additional information may be found in the Windows Event Logger. In the right-hand panel, Find the event associated with the installation failure. Double-click the event to display the details of the failure.

Accessing the Application

If the application includes the ASP.NET Web Clients, the application can be accessed by using Microsoft Internet Explorer and supplying a URL using the following format:

```
<host>[:CA Portal]/<app-name>/<trancode>.ashx
```

For example:

```
localhost/myapp/menu.ashx
```

Chapter 6: Diagram Tracing .NET Servers

Running Diagram Trace on CA Gen .NET Servers requires the generated Server Procedure Step to connect to a Diagram Trace Utility executing on a Windows System.

To run Diagram Trace on a CA Gen .NET Server,

Follow these steps:

1. Generate the Procedure Steps and Action Blocks with Trace.
2. Specify the Diagram Trace Utility host and port number to the servers.
3. Start the Diagram Trace Utility.
4. Run the application to be traced.

Generating the Code

There are two ways to generate the .NET Server for Trace:

- Specify Generate source code with Trace in the Generation Options dialog.
- Check the TRCE flag for all code to be traced.

Specifying the Trace Server

CA Gen .NET Servers do not have a user interface. Therefore, interacting with the Diagram Trace dialogs requires the use of the Diagram Trace Utility. This program executes on a Windows system. The .NET Server connects to the Diagram Trace Utility using TCP/IP.

You can specify the Diagram Trace Utility information during the assemble process or afterwards by modifying the application's config files.

Setting During the Assemble Process

The Diagram Trace Utility host and port can be specified during the assemble process.

Follow these steps:

1. On the MSI .NET Assemble Details dialog, select Tracing from the tree view on the left.
2. In the details area on the right, select the Enable Diagram Tracing check box.

3. Specify the host and port number of the Diagram Trace Utility in the fields provided. The default values are:

   Trace Server Name: localhost

   Trace Server Port: 4567

   Note: The Trace Server Name may be an IPv4 or IPv6 address.

Setting After Installation

The Diagram Trace Utility's host and port can be specified after the application has been installed by editing the web.config and/or the application.config file:

Follow these steps:

1. Navigate to the application directory, the directory named after the application under the IIS wwwroot directory. For example, the application directory for the application IssueTracking is C:\Inetpub\wwwroot\IssueTracking.
2. For the CA Gen ASP.NET Web clients, edit the web.config file. For the CA Gen .NET Servers, edit the application.config file in the bin subdirectory.
3. Search for the string Tracing. The following text appears:

   ```
   <add key="Tracing" value="false" />
   <add key="TraceHost" value="localhost" />
   <add key="TracePort" value="4567" />
   ```

   Note: localhost can be an IPv4 or IPv6 address.
4. Change the value of Tracing to True to enable Diagram Tracing. If necessary, modify the host and port specification of the Diagram Trace Utility.

Starting the Diagram Trace Utility

You may start the Diagram Trace Utility by using the Start menu. Click Start, All Programs, CA, Gen *xx*, Diagram Trace Utility.

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

The Diagram Trace Utility will "autostart" and listen on port 4567. You may change the default port from within the Diagram Trace Utility.

Note: For more information on the Diagram Trace Utility, see the *Diagram Trace Utility User Guide*.

Running the Application with Trace

Once the Diagram Trace Utility is running and ready to accept connections, the CA Gen .NET Servers can be started. Access the CA Gen .NET Server as it would normally be accessed. The connection to the Diagram Trace Utility takes place shortly after the CA Gen .NET Server has been initialized.

Chapter 7: Cooperative Flows

This chapter details the cooperative flows between generated clients and servers implemented in different languages. These include:

- CA Gen MFC GUI Clients as well as CA Gen C Proxy on Windows and COM Proxy Clients to CA Gen .NET Servers using the CA Gen C to C# Cooperative Flow.
- CA Gen .NET Servers to CA Gen C and COBOL Servers using the CA Gen TCP/IP or MQSeries Cooperative Flows.

CA Gen C to C# Cooperative Flow

The CA Gen-generated C language Clients are:

- CA Gen MFC GUI Clients
- CA Gen C Proxy Clients on Windows
- CA Gen COM Proxy Clients

These clients use the CA Gen C to C# Cooperative Flow to communicate with the CA Gen .NET Servers. This cooperative runtime performs the data conversion and .NET Remoting call on the client system.

Prerequisites

The CA Gen-generated C language clients must have been generated and built with CA Gen Release 7 or later and use the same version of the cooperative runtimes.

The client system must have the Microsoft .NET Framework installed to execute the cooperative flow.

Configuration

There are a number of steps that must be performed to have the CA Gen-generated C language clients interoperate with the CA Gen .NET Servers, as described later.

Configuring the Cooperative Flow

Configuring the cooperative flow can be accomplished by either one of the two following methods:

- Change or define the server properties for .NET Remoting and regenerate, ensure the server environment parameters are set to the .NET environment and regenerate the CA Gen-generated C language Clients
- Modify the COMMCFG.INI file to specify the new target CA Gen .NET Server.

Each of these options is described in detail.

Option 1: Change Your Model and Generate

During generation, the generated clients know the server they will be using at runtime and the default cooperative flow to use for that server. The cooperative flow information comes from the server properties dialog and server environment parameters dialog.

Follow these steps:

1. Open your model. On the main menu bar, select Construction, Packaging or Construction, Generation.
2. Open the cooperative packaging diagram by selecting Diagram, Open, Cooperative Code.
3. For each server that is generated for the .NET environment:
    a. Open the server manager properties dialog by either double-clicking the server, select the server and then selecting Details, Properties from the main menu bar or right-clicking the server and selecting Detail, Properties from the pop-up menu.
    b. In the server manager dialog, select the .NET Remoting tab.
    c. Enter values in to the HostName and Service (Port) fields. HostName is the TCP/IP hostname where the CA Gen-generated .NET Servers are executing. The Service (Port) is the port number of IIS, which is usually 80. For the best performance the channel type should be Binary.
    d. Click OK to save your changes.
    e. With the server manager still selected, open the server environment parameters dialog by either selecting Detail, Server Environment from the main menu bar or right-clicking on the server and selecting Detail, Server Environment from the pop-up menu.
    f. Set the server Operating System to CLR and Communications to .NET_Remoting.
    g. Click OK to save your changes.

4. After all servers have been configured, generate and build the CA Gen-generated C language clients. The client now contains data that defaults to interoperating with the CA Gen-generated .NET Servers.
5. Prior to executing the client, ensure the COMMCFG.INI file does not contain an entry that will override the information generated in the client.

Option 2: Modifying COMMCFG.INI

Override the cooperative flow information generated in the client by editing the COMMCFG.INI file.

Follow these steps:

1. In the editor of your choice, open the file <Gen-Root-Dir>\COMMCFG.INI:

   ```
   cd %GENxx%Gen
   write commcfg.ini
   ```

2. Create an entry for the trancode(s) to be redirected to the CA Gen-generated .NET Servers:

   ```
   <trancode> NET <host> CA Portal B
   ```

   where <host> is the TCP/IP hostname where the CA Gen-generated .NET Server is executing and CA Portal is the port number of IIS (usually 80).

Note: For a detailed discussion on the use of the COMMCFG.INI file, see the *Distributed Processing – Overview Guide*.

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Copy CA Gen Files to the Client Application's Directory

In order for the client application to execute the CA Gen C to C# Cooperative Flow, several files must be copied to the client application's directory. Copy the following files from the CA Gen directory structure to the client application's directory.

From the CA Gen root directory:

- CA.Gen.odc.c2cs.dll

From the .net\bin directory under the CA Gen root directory:

- CA.Gen.csu.dll
- CA.Gen.exits.dll
- CA.Gen.odc.complus.dll
- CA.Gen.odc.dll

- CA.Gen.odc.net.dll
- CA.Gen.vwrt.dll

For example:

```
cd <application directory>
copy "%GENxx%Gen\CA.Gen.odc.c2cs.dll"
copy "%GENxx%Gen\.net\bin\CA.Gen.csu.dll"
copy "%GENxx%Gen\.net\bin\CA.Gen.exits.dll"
copy "%GENxx%Gen\.net\bin\CA.Gen.odc.complus.dll"
copy "%GENxx%Gen\.net\bin\CA.Gen.odc.dll"
copy "%GENxx%Gen\.net\bin\CA.Gen.odc.net.dll"
copy "%GENxx%Gen\.net\bin\CA.Gen.vwrt.dll"
```

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Copy Files from the .NET Server Build Machine

In order for the client system to access the CA Gen .NET Servers on the target system, the portable interface definition files must be copied to the client application's directory.

During the construction of the generated .NET Servers, a file (one per load module) containing the generated .NET Server's portable interface definitions is created. The file is named <load-module>.interfaces.dll and is located in the <model-root>\c#\build directory.

For the servers to be accessed, copy the <load-module>.interfaces.dll file(s) from the <model-root>\c#\build directory to the client application's directory.

For example:

```
cd <application directory>
copy "<model-root>\c#\build\<load-module>.interfaces.dll"
```

Execute the Application

After all the above tasks have been completed, execute the CA Gen-generated C language client as before.

CA Gen TCP/IP Cooperative Flow to CA Gen C and COBOL Servers

The CA Gen .NET Servers may interoperate with CA Gen C and COBOL Servers using the CA Gen TCP/IP Cooperative Flow.

Configuration

There are a few steps that must be performed to have the CA Gen .NET Servers interoperate with the CA Gen C and COBOL Servers, as described in the following sections.

Configuring the Cooperative Flow

Configuring the cooperative flow can be accomplished by either one of the two following methods:

- Change or define the server properties for TCP/IP and regenerate, ensure the server environment parameters are set to the C or COBOL environment with the correct communications setting and regenerate the CA Gen .NET Servers

OR

- Modify the COMMCFG.TXT file to specify the new target CA Gen C or COBOL Server.

A detailed description of each option follows.

Option 1: Change Your Model and Generate

During generation, the generated servers know the servers they will be utilizing at runtime and the default cooperative flow to use for that server. The cooperative flow information comes from the server properties dialog and server environment parameters dialog.

Follow these steps:

1. Open your model. On the main menu bar, select Construction, Packaging or Construction, Generation.
2. Open the cooperative packaging diagram by selecting Diagram, Open, Cooperative Code.
3. For each server that is generated for either C or COBOL:
    a. Open the server manager properties dialog by either double-clicking the server, select the server and then selecting Details, Properties from the main menu bar or right-clicking the server and selecting Detail, Properties from the pop-up menu.
    b. In the server manager dialog, select the TCP/IP tab.
    c. Enter values in to the HostName and Service (Port) fields. HostName is the TCP/IP hostname where the CA Gen C or COBOL Servers are executing. The Service (Port) is the port number of the TP Monitor running the CA Gen C or COBOL Servers.
    d. Click OK to save your changes.
    e. With the server manager still selected, open the server environment parameters dialog by either selecting Detail, Server Environment from the main menu bar or right-clicking the server and selecting Detail, Server Environment from the pop-up menu.
    f. Set the server Operating System as appropriate and Communications to TCP/IP.
    g. Click OK to save your changes.
4. When all servers have been configured, generate and build the CA Gen .NET Servers. The servers now contain the data needed to interoperate with the CA Gen C or COBOL Servers.
5. Ensure the COMMCFG.TXT file does not contain an entry that overrides the information generated in the servers.
6. Assemble the servers into an MSI file and install the servers on the target system.

Option 2: Modifying COMMCFG.TXT

CA Gen .NET Servers can be redirected to CA C and COBOL Servers by overriding the cooperative flow information generated in the server as a result of editing the COMMCFG.TXT file.

Follow these steps:

1. Locate the file to be edited. The COMMCFG.TXT file is located under the CA Gen directory in .net\bin. After installation, all applications have a copy of this file in the \Inetpub\wwwroot\<app-name>\bin directory. This file may also be directly edited.

   Editing the COMMCFG.TXT file under the CA Gen directory requires that the application be reassembled into an MSI file and reinstalled.

2. In the editor of your choice, open the COMMCFG.TXT file

   ```
   write commcfg.txt
   ```

3. Create an entry for the trancode(s) to be redirected to the CA Gen generated .NET Servers, for example:

   <trancode> = TCP <host> CA PortalTCP <host> CA Portal

4. If necessary, reassemble the MSI file and reinstall the application. This should only be done if the COMMCFG.TXT file under the CA Gen directory has been edited.

Note: For a detailed discussion on the use of the COMMCFG.TXT file, see the *Distributed Processing – Overview Guide*.

Execute the Application

After all the above tasks have been completed, execute the application as before.

CA Gen MQSeries Cooperative Flow to CA Gen C and COBOL Servers

The CA Gen .NET Servers may interoperate with CA Gen C and COBOL Servers using the CA Gen MQSeries Cooperative Flow.

Configuration

There are a few steps that must be performed to have the CA Gen .NET Servers interoperate with the CA Gen C and COBOL Servers, as described in the following sections.

Configuring the Cooperative Flow

Configuring the cooperative flow can be accomplished by either one of the two following methods:

- Change or define the server properties for MQSeries and regenerate, ensure the server environment parameters are set to the C or COBOL environment with the correct communications setting and regenerate the CA Gen .NET Servers

OR

- Modify the COMMCFG.TXT file to specify the new target CA Gen C or COBOL Server.

A detailed description of each option follows.

Option 1: Change Your Model and Generate

During generation, the generated servers know the servers they will be utilizing at runtime and the default cooperative flow to use for that server. The cooperative flow information comes from the server properties dialog and server environment parameters dialog.

Follow these steps:

1. Open your model. On the main menu bar, select Construction, Packaging or Construction, Generation.
2. Open the cooperative packaging diagram by selecting Diagram, Open, Cooperative Code.
3. For each server that is generated for either C or COBOL:
    a. Open the server manager properties dialog by either double-clicking the server, select the server and then selecting Details, Properties from the main menu bar or right-clicking the server and selecting Detail, Properties from the pop-up menu.
    b. In the server manager dialog, select the MQSeries tab.
    c. Enter values in Queue Manager Name, Queue Name, Reply Queue Name, and Client Type fields.
    d. Click OK to save your changes.
    e. With the server manager still selected, open the server environment parameters dialog by either selecting Detail, Server Environment from the main menu bar or right-clicking the server and selecting Detail, Server Environment from the pop-up menu.
    f. Set the server Operating System as appropriate and Communications to MQSeries.
    g. Click OK to save your changes.
4. When all servers have been configured, generate and build the CA Gen .NET Servers. The servers now contain the data needed to interoperate with the CA Gen C or COBOL Servers.
5. Ensure the COMMCFG.TXT file does not contain an entry that overrides the information generated in the servers.
6. Assemble the servers into an MSI file and install the servers on the target system.

Option 2: Modifying COMMCFG.TXT

CA Gen .NET Servers can be redirected to CA C and COBOL Servers by overriding the cooperative flow information generated in the server as a result of editing the COMMCFG.TXT file.

Follow these steps:

1. Locate the file to be edited. The COMMCFG.TXT file is located under the CA Gen directory in .net\bin. After installation, all applications have a copy of this file in the \Inetpub\wwwroot\<app-name>\bin directory. This file may also be directly edited.

   Editing the COMMCFG.TXT file under the CA Gen directory requires that the application be reassembled into an MSI file and reinstalled.

2. In the editor of your choice, open the COMMCFG.TXT file

   ```
   write commcfg.txt
   ```

3. Create an entry for the trancodes to be redirected to the CA Gen generated .NET Servers, for example:

   <trancode> = MQS {Queue Manager} {Queue Name} {Reply Queue Name}

4. If necessary, reassemble the MSI file and reinstall the application. This should only be done if the COMMCFG.TXT file under the CA Gen directory has been edited.

Note: For a detailed discussion on the use of the COMMCFG.TXT file, see the *Distributed Processing – Overview Guide*.

Execute the Application

After all the above tasks have been completed, execute the application as before.

CA Gen Web Services Cooperative Flow to CA Gen EJB or TE Servers

The CA Gen .NET Servers may interoperate with CA Gen EJB or TE Web Service Servers using the CA Gen Web Services Cooperative Flow.

Configuration

There are a few steps that must be performed to have the CA Gen .NET Servers interoperate with the CA Gen EJB or TE Web Service Servers, as described in the following sections.

Configuring the Cooperative Flow

Configuring the cooperative flow can be accomplished by modifying the COMMCFG.TXT file to specify the new target CA Gen EJB or TE Web Service Server.

A detailed description follows.

Modifying COMMCFG.TXT

CA Gen .NET Servers can be redirected to CA EJB Web Service Servers by overriding the cooperative flow information generated in the server as a result of editing the COMMCFG.TXT file.

1. Locate the file to be edited. The COMMCFG.TXT file is located under the CA Gen directory in .net\bin. After installation, all applications have a copy of this file in the \Inetpub\wwwroot\<app-name>\bin directory. You can edit this file directly. Editing the COMMCFG.TXT file under the CA Gen directory requires that the application be reassembled into an MSI file and reinstalled.
2. Open the COMMCFG.TXT file in an editor.

   ```
   write commcfg.txt
   ```

3. Create an entry for the trancodes to be redirected to the CA Gen generated .NET Servers, for example:

   ```
   <trancode> = WS <baseURL> <contextType>
   ```

4. Reassemble the MSI file and reinstall the application if necessary. Do this task only if you edited the COMMCFG.TXT file under the CA Gen directory.

Note: For a detailed discussion about the use of the COMMCFG.TXT file, see the *Distributed Processing – Overview Guide.*

Execute the Application

After all the above tasks have been completed, execute the application as before.

Chapter 8: User Exits

User exits allow you to change the behavior of the CA Gen runtimes. User exits are C# classes that are shipped with the product and contain specific routines that are called by the runtimes. You can modify the source code to change the default behaviors of the runtimes.

All C# user exits are located in subdirectories under the <CAGen-root>\.net\exits\src directory. For example:

```
cd %GENxx%Gen
cd .net\exits\src
```

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Note: For a detailed discussion of many of the user exits listed in this chapter, see the *User Exit Reference Guide*.

The Exits

The following exits are located in the <CAGen-root>\.net\exits\src\amrt directory and used by the CA Gen ASP.NET Web Clients:

- DefaultYearExit.cs
- LocaleExit.cs
- RetryLimitExit.cs
- SessionIdExit.cs
- SrvrErrorExit.cs
- UserExit.cs

The following exits are located in the <CAGen-root>\.net\exits\src\coopflow directory and used by both the CA Gen ASP.NET Web Clients and CA Gen .NET Servers:

- COMPLUSDynamicCoopFlowExit.cs
- COMPLUSDynamicCoopFlowSecurityExit.cs
- NETDynamicCoopFlowExit.cs

- NETDynamicCoopFlowSecurityExit.cs
- TCPIPDynamicCoopFlowExit.cs
- MQSDynamicCoopFlowExit.cs
- WSDynamicCoopFlowExit.cs

The following exits are located in the <CAGen-root>\.net\exits\src\msgobj directory and used by both the CA Gen ASP.NET Web Clients and CA Gen .NET Servers:

- CFBDynamicMessageDecryptionExit.cs
- CFBDynamicMessageEncodingExit.cs
- CFBDynamicMessageEncryptionExit.cs
- CFBDynamicMessageSecurityExit.cs

The following exits are located in the <CAGen-root>\.net\exits\src\scrt directory and used by the CA Gen .NET Servers:

- AuthorizationExit.cs
- LocaleExit.cs
- RetryLimitExit.cs
- SecurityValidationExit.cs
- SrvrErrorExit.cs
- UserExit.cs

The following exits are located in the <CAGen-root>\.net\exits\src\Common.cs file used by CA Gen ASP.NET Web Clients and CA Gen .NET Servers:

- CompareExit
- LowerCaseExit
- UpperCaseExit

After the desired user exit source file has been located, make a backup copy of the file, then edit the file and save it back to its original name and location. For detailed information about the purpose and use of each method, see the comments in the exit source.

Important! Do not modify the namespace name. Changing the namespace name will cause the exit not to be recognized at runtime.

Since the user exit source files are saved to their original locations, it may be useful to save the original and modified sources in a configuration management system to prevent the changes from being lost should a new installation overwrite the source file.

Redistributing the Exits

After the exits have been compiled, they are ready for redistribution. The procedure for redistribution depends on the type of application and the way the original exits were distributed.

- For CA Gen ASP.NET Web Clients or CA Gen .NET Servers, the action depends on whether the MSI file contains a copy of the CA Gen runtime. If the MSI file was assembled with a copy of the CA Gen runtime, then the application should be re-assembled and reinstalled. This causes all updated exits to be pulled into the runtime DLL file and inserted into the MSI file.
- If the runtime is being distributed separately from the MSI files, or if it is being used in a C# Proxy, then the original distribution procedures must be repeated.

Chapter 9: External Action Blocks

External Action Blocks (EABs) allow your CA Gen application to access logic that is not modeled in or generated by CA Gen.

An external action block stub is a code skeleton created when you generate an EAB using a Construction toolset. They are designed to be generated once and then modified and maintained outside of CA Gen. You are responsible for writing and compiling the external subroutine and making it available to the application.

The stub contains the definitions necessary for the skeleton to be called by other generated action blocks and comments that indicate where user code may be added. You may either add a call to other classes and methods or you may add the logic directly into the EAB stub.

This chapter describes the tasks related to External Action Blocks and CA Gen C#-generated applications.

.NET Environment Requirements

In the .NET environment, source code is compiled and linked into an assembly in a single step. Consequently, the CA Gen Build Tool does not compile the EAB stubs since they would be included in the application's ready-for-execution assembly.

A model with EABs that have not been separately compiled and identified to the CA Gen Build Tool will fail compilation with errors similar to the following:

```
CEXIT.cs(54,9): error CS0234: The type or namespace name 'DEMOEAB_IA' does not exist
in the class or namespace 'com.ca' (are you missing an assembly reference?)

CEXIT.cs(55,9): error CS0234: The type or namespace name 'DEMOEAB_OA' does not exist
in the class or namespace 'com.ca' (are you missing an assembly reference?)
```

These errors indicate the Import and Export views of the DEMOEAB have not been compiled and made available to the C# compiler.

In other environments, these errors would not occur until runtime where an invalid address or class not found error would be encountered.

User Action Outside of CA Gen

Copy the generated EAB stub and associated view files (<class>_IA, <class>_OA) to a separate location before modifying the stub. This prevents the stub, and your code, from being overwritten when the EAB is regenerated. The generated stub is stored in the C# directory under the model directory.

Add the logic needed to meet the specific requirements of your application. You may use the modified stub to call an existing external subroutine or you may include the subroutine logic in the stub.

Accessing Individual Class Members

The import and export views are placed into separate class files. For example, the action block DEMO_EAB will be generated as the class DEMOEAB. Its import and export views are generated as the classes DEMOEAB_IA and DEMOEAB_OA, respectively.

Within each class, the individual class members hold the view data. Views, entities, and attribute names are concatenated together and generated as a single flattened variable, without underscores for spacing. The single variable may contain non-ASCII characters since C# allows them in variable names. The variable follows the uppercase and lowercase rules typically used in C# programming. Specifically, the first letter of each word is in uppercase and all remaining characters are in lowercase.

For example:

Entity View: IMPORT_CUSTOMER
Entity: ADDRESS
Attribute: POSTAL_CODE

becomes

ImportCustomerAddressPostalCode

Each member of the class may be defined as one of the following native C# data types:

char
short
int
double

Further, some members of the class are defined as one of the following native C# classes:

decimal
string

Within a view, each attribute is generated as a .NET property which means that it may be accessed as a standard C# property on an object.

For example:

```
int Id = WIa.ImportCustomerAddressId;
WOa.ExportCustomerAddressId = Id;
```

Compiling the EABs

The CSC command (C# compile) is used to compile the EAB into either a .NET module or assembly. For EABs compiled as a .NET module, the CA Gen Build Tool builds the generated application and includes the .NET modules in the application's assemblies. For EABs compiled as assemblies, the CA Gen Build Tool builds the generated application and references the EABs in the assembly. During the assemble process, the assembly containing the EABs must be specified on the CA Gen Build Tool's assemble dialog so that the assembly is included in the MSI.

.NET Modules

This section explains what actions to take in the Build Tool to compile External Action Blocks as .NET modules.

Module Info File

Microsoft defines CLS as a set of language semantics that allows any .NET aware compiled language to interoperate with any other .NET aware compiled language. CA Gen-generated C# code is compiled as CLS compliant into assemblies using the assemblyinfo.cs file. For C# code to be compiled into a .NET module, a moduleinfo.cs file needs to be created and used as input to the compiler.

Create a file named moduleinfo.cs, which contains the following text:

```
using System; using System;
[module: CLSCompliantAttribute(true)]
```

There are several more module-level attributes that may be set.

Note: For more information, see the Microsoft documentation.

Compilation

To compile the EABs into a .NET module, use a CSC command similar to the following:

```
csc /target:module /out:eab.netmodule "/lib:%GENxx%Gen\.net\bin"
/reference:CA.Gen.abrt.dll,CA.Gen.csu.dll,CA.Gen.exits.dll,CA.Gen.vwrt.dll,CA.Gen
.odc.dll moduleinfo.cs EAB1*.cs EAB2*.cs EAB3*.cs …
```

where:

/target:module specifies that the result of the compilation is a .NET module (.netmodule)

/out:eab.netmodule specifies the name of the output module (EAB.NETMODULE)

"/lib:%GENxx%Gen\.net\bin" specifies the directory where the assemblies to be referenced during the compilation reside.

Note: The surrounding quote marks are required as the IEFH environment variable typically contains spaces.

/reference:CA.Gen specifies a comma separated list of assemblies to be referenced during the compilation.

moduleinfo.cs is the file containing the CLS-compliant module attribute.

EAB1*.cs EAB2*.cs EAB3*.cs ... specifies a space separated list of C# source files to be compiled. The asterisk is used to cause the EAB file plus its import and export view class files to be compiled as a single unit.

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Build Tool

In the CA Gen Build Tool, the .NET module file name (with its directory) must be specified within your current profile and entered through the profile manager.

Follow these steps:

1. Open the Profile Manager and specify the profile to modify.
2. In the tree view on the left, select NET.
3. In the properties grid on the right, change the token value LOC.NET_EXTERNAL_MODULES to specify the external modules to be added into the generated assemblies. For example c:\myEABs\eabs.dll

Assemble

Since the EABs compiled as .NET modules are included in the application assemblies, no additional information must be specified on the MSI .NET Assemble Details dialog.

Assemblies

This section explains what actions to take in the Build Tool to compile External Action Blocks into an assembly.

Strong Naming

Microsoft introduced strong naming to improve the runtime security of the application. Assemblies created by CA Gen are strongly named. Therefore, the assembly containing the EABs must also be strongly named.

Use the following procedure to create a strongly named assembly:

Follow these steps:

1. Create the key file to be referenced by assemblyinfo.cs. Click Start, All Programs, Microsoft Visual Studio 2010, Visual Studio Tools, Visual Studio Command Prompt (2010). At the command prompt, enter the following command:

   ```
   sn -k eab.snk
   ```

2. Create an assemblyinfo.cs file that contains the following text:

   ```
   using System;
   using System.Reflection;

   [assembly: AssemblyCompanyAttribute ("your company name")]
   [assembly: AssemblyDescriptionAttribute ("a description")]

   [assembly: AssemblyVersionAttribute ("1.0.0.0")]
   [assembly: AssemblyKeyFileAttribute ("eab.snk")]
   [assembly: CLSCompliantAttribute(true)]
   ```

   The only required entries are AssemblyKeyFileAttribute and CLSCompliantAttribute. There are several more assembly-level attributes that may be set.

   Note: For more information, see the Microsoft documentation.

   These two files will be used during compilation.

Compilation

To compile the EABs into an assembly, use a CSC command similar to the following:

```
csc /target:library /out:eab.dll "/lib:%GENxx%Gen\.net\bin"
/reference:CA.CAGen.abrt.dll,CA.CAGen.csu.dll,CA.CAGen.exits.dll,CA.CAGen.vwrt.dl
l,CA.CAGen.odc.dll assemblyinfo.cs EAB1*.cs EAB2*.cs EAB3*.cs …
```

where:

/target:library specifies that the result of the compilation is an assembly (DLL)

/out:eab.dll specifies the name of the output assembly (EAB.DLL)

"/lib:%GENxx%Gen\.net\bin" specifies the directory where the assembly to be referenced during the compilation reside.

Note: The surrounding quote marks are required as the IEF environment variable typically contains spaces.

/reference:CA.Gen... specifies a comma separated list of assemblies to be referenced during the compilation

assemblyinfo.cs is the file containing the assembly attributes.

EAB1*.cs EAB2*.cs EAB3*.cs ... specifies a space separated list of C# source files to be compiled. The asterisk is used to cause the EAB file plus its import and export view class files to be compiled as a single unit.

You may see a warning similar to the following:

```
DEMOEAB.cs(1,11): warning CS3012: You must specify the CLSCompliant attribute on the
assembly, not the module, to enable CLS compliance checking
```

This is caused by the presence of the module attribute at the top of the source file to set the CLSCompliant attribute to true. For assemblies, this setting is unnecessary. The warning may be safely ignored or the module attribute setting may be commented out.

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Build Tool

In the CA Gen Build Tool, the directory and assembly file name must be specified within your current profile and entered through the profile manager.

Follow these steps:

1. Open the Profile Manager and specify the profile to modify.
2. In the tree view on the left, select NET.
3. In the properties grid on the right, change the following tokens:
    a. LOC.NET_EXTERNAL_LIB_DIRECTORIES specifies the directories where the external assemblies are located.
    b. LOC.NET_EXTERNAL_ASSEMBLIES specifies the file names of the external assemblies to be referenced during the C# compile.

Assemble

During the assemble process, you must specify the location and name of the assembly or assemblies containing the EABs to be included in the MSI file.

Note: For more information on the assemble process, see the *Build Tool Guide*.

Chapter 10: Component Based Development

In component-based development (CBD), logic for an application can be separated into discrete functional elements or components.

Each component is represented by an implementation model and a specification model. An implementation model contains the actual programming logic of the component. The specification model contains the interface definition for the component. Specifically, this model contains the procedure step and action block signatures as well as data model elements needed for users of the component.

The final player in CBD is the consuming model. This model imports the specification model and uses the interface definitions to make calls to the components. During the assemble process, the executable portions of the implementation model are combined with the built consuming model to create a complete application. A consuming model may consume more than one component. The consuming model could be a component and be consumed.

Each public operation in the component may be transactional or subtransactional in nature. Transactional public operations are modeled as procedure steps. Subtransactional public operations are modeled as action blocks in the implementation model and external action blocks in the specification model.

This chapter discusses additional topics of interest to customers who are using CBD techniques.

Restrictions

The consumption of UI-bearing components (client procedure steps) is not supported at this time.

Setting Namespace Names

To allow components to be easily consumed, the following namespace name settings are recommended:

- In the implementation model, define a Namespace Name at the Model Level
- In the consuming model, import the spec model into its own Business System and set its namespace name to that of the implementation model

Consuming Subtransactional Components

Subtransactional public operations are modeled as action blocks in the implementation model and external action blocks in the specification model.

The CA Gen Build Tool does not compile the external action block. To successfully *compile* the consuming application, the .NET modules or assemblies must be made available to the CA Gen Build Tool.

The actions to take in the CA Gen Build Tool depend upon whether the subtransactional components are delivered as .NET modules or assemblies.

.NET Modules

This section explains the actions to be taken in the Build Tool when the subtransactional components are delivered as .NET modules.

Build Tool

In the Build Tool, the.NET module file name (with its directory) must be specified within your current profile and entered through the profile manager.

Follow these steps:

1. Open the Profile Manager and specify the profile to modify.
2. In the tree view on the left, select NET.
3. In the properties grid on the right, change the token value LOC.NET_EXTERNAL_MODULES to specify the external modules to be added into the generated assemblies.

Assemble

Since the subtransactional component was compiled as .NET modules, they are included in the application assemblies. No additional information must be specified on the MSI .NET Assemble Details dialog.

Assemblies

This section explains the actions to be taken in the Build Tool when the subtransactional components are delivered as .NET modules.

Build Tool

In the CA Gen Build Tool, the directory and assembly file name must be specified within your current profile and entered through the profile manager.

Follow these steps:

1. Open the Profile Manager and specify the profile to modify.
2. In the tree view on the left, select NET.
3. In the properties grid on the right, change the following tokens:
    a. LOC.NET_EXTERNAL_LIB_DIRECTORIES specifies the directories where the external assemblies are located.
    b. LOC.NET_EXTERNAL_ASSEMBLIES specifies the file names of the external assemblies to be referenced during the C# compile.

Assemble

During the assemble process, you must specify the location and name of the assembly or assemblies containing the subtransactional public operations to be included in the MSI file.

Note: For more information on the Additional Files panel in the assembly dialog, see the *Build Tool Guide*.

Consuming Transactional Components

Transactional public operations are modeled as procedure steps.

In the consuming model, the transactional public operations should be located in separate business systems. *Do not* generate these procedure steps.

During the assemble process, you must specify the location and name of the assembly or assemblies containing the transactional public operations to be included in the MSI file.

Note: For more information on the Additional Files panel in the assembly dialog, see the *Build Tool Guide*.

Chapter 11: Handcrafted Clients

Customers can access CA Gen .NET Servers from handcrafted clients. The goal is for the CA Gen .NET Servers to be as easy to access as handcrafted .NET Servers. To that end, the CA Gen .NET Servers are called using .NET Remoting.

View Definitions

CA Gen .NET Servers have an import and export view that send data to and from the server respectively. The following sections describe the data representation and layout of the generated servers.

View Objects

The view data is generated into Import and Export Objects. Some of the benefits of the functionality built into the view objects are:

- Comprehensive—View data and system data together.
- Hierarchical—More intuitive access paths to the data.
- Object-oriented—Allows more sophisticated manipulation of the data.
- Serializable—Allows object state to be saved and restored.
- No Runtime—Enables easier transmission/distribution/serialization.
- Data Validation—(Immediate) Keeps view always in a consistent state.
- Cloneable—Built in support for cloning.
- Resettable—Multi-level reset ability gives greater control.
- GroupView Support—Allows row-level operations or manipulations.

Import Objects

The Import objects are generated in classes called <method-name>Import based on the import view designed in the model for the CA Gen .NET Server. It contains the actual import data for the execution of the server. The Import objects are also responsible for all data validation that is performed on the attributes of the import views.

Even if the server does not contain any import views, the Import object is still generated because the system level data (Command, NextLocation, and so on) are always present. The developer must instantiate an Import object and populate it with the data. The instance is then passed onto the execution methods as a parameter.

The Import objects are stateful. In fact, they exist to hold a state. Therefore, the developer must be careful not to use a particular instance of one of these classes between different threads in a multi-threaded application.

Export Objects

The Export objects are generated in classes called <method-name>Export based on the export view designed in the model for the CA Gen .NET Server. It contains the actual export data for the execution of the server. The Export objects are also responsible for all data validation that is done on the attributes of the export views.

Even if the server does not contain any export views, the Export object is still generated because the system level data (Command, ExitState, and so on) are always present. The server object execution methods create and populate the Export objects.

The Export objects are stateful. In fact, they exist to hold a state. Therefore, the developer must be careful not to use a particular instance of one of these classes between different threads in a multi-threaded application.

View Example

The easiest way to understand view objects is to look at an example. Later sections in this chapter describe how particular mappings occur.

The following example shows how generators map a given import view in the model to a view object.

```
ADDR_SERVER

IMPORTS:
   Entity View in address
       line1
       line2
   Group View inGV (2)
       Entity View ingroup address
           line1
           line2
```

From the given import view, five .NET classes are generated. The classes and their properties are shown in the following table:

Class	Attributes
AddrServerImport	■ Command ■ NextLocation ■ ClientId ■ ClientPassword ■ Dialect ■ ExitState ■ InEVAddress—Gets AddrServer.InAddress object ■ IngvGV—Gets AddrServer.Ingv object
AddrServer.InAddress	■ Line1Fld ■ Line2Fld
AddrServer.Ingv	■ Capacity (Read-only constant) ■ Length ■ Rows—Gets a AddrServer.IngvRow object ■ this[]—Implicit indexer same as Rows
AddrServer.IngvRow	■ IngroupEVAddress—Gets AddrServer.IngroupEVAddress object
AddrServer.IngroupAddress	■ Line1Fld ■ Line2Fld

The number of classes increases in proportion with the number of entity views, work sets, and group views.

To finish the sample, it is beneficial to look at how a programmer gets and sets the various pieces of data in the views. The following code fragment is written in C#, but the equivalent VB.NET code would be very similar.

Instantiate the Import View object:

```
AddrServerImport importView = new AddrServerImport();
```

Set the command system level data:

```
importView.Command = "SEND";
```

Access the InEVAddress line2 attribute:

```
String value = importView.InEVAddress.Line2Fld;
```

Get the maximum capacity of the IngvGV group view:

```
for (int i = 0; i < importView.IngvGV.Capacity; i++)
```

Set the current IngvGV number of rows:

```
importView.IngvGV.Length = 2;
```

Set the line1 attribute of the IngroupEVAddress entity view:

```
importView.Ingv[2].IngroupEVAddress.Line1Fld = "ABC";
```

Reset the InAddress Entity View back to defaults:

```
importView.InAddress.Reset();
```

System Level Properties

The import and export objects contain a set of properties at their top level that is best described as system-level properties. Not all the servers make use of these properties, but they are always present on each cooperative server call. The following table shows the properties of import and export objects:

Objects	Properties
Import Objects	■ Command ■ NextLocation ■ ExitState ■ Dialect ■ ClientId ■ ClientPassword
Export Objects	■ Command ■ ExitState ■ ExitStateType ■ ExitStateMessage

GroupView Objects

Special objects are generated for each group view. The first is the group view object itself. This object holds the Capacity or the maximum number of rows specified in the model. It also holds the Length or the current number of populated rows in the group view. The Length cannot exceed the Capacity or be negative. If you try to do so, it throws IndexOutOfRangeException or ArgumentOutOfRangeException.

To store row data, an object is generated. It represents a row and contains references to all the entity and work set views within the GroupView. This group view object then stores an array of row objects to hold the data. An individual row can be accessed by indexing the Rows property on the group view, or by using the indexer property built into the group view object itself.

Since a row of data is an object, it allows certain row-level operations to be performed, such as looping, cloning, and resetting.

Data Validation

The import and export views validate their attribute data when it is set. The validation checks performed throws ArgumentExceptions if the new value is not valid. The views perform the following validation checks: permitted values, length/precision, and mandatory/optional.

Default Values

Attribute view data in the import and export objects enforce the default values, as defined in the model. The default value is applied when the object is created or the attribute is reset programmatically.

Data Type Mappings

Since the import and export objects are designed to have no runtime dependencies, the attribute data types must be native data types supported by the .NET Framework. The following table shows the mappings applied:

CA Gen Attribute Definition	.Net Framework Data Type
Text/Mixed Text/DBCS Text	System.String
Number (no decimals, =< 4 digits)	short
Number (no decimals, =< 9 digits)	int
Number (no decimals, > 9 digits)	double

Number (with decimals)	double
Number (with precision turned on)	System.Decimal
Date	System.DateTime (null represents optional data)
Time	System.DateTime (null represents optional data)
Timestamp	System.DateTime (null represents optional data)

Coding the Invocation

The following code would be used to invoke the .NET server:

```
...
<method-name>Import importView = new <method-name>Import();
<method-name> server = new <method-name>();
...
<define the importView properties as desired>
...
// In its simplest form:
 <method-name>Export exportView = server.Execute(importView, null, null);

// In its complete form:
 <method-name>Export exportView = server.Execute(importView, securityObject,
userObject);
...
<retrieve exportView data>
...
```

The securityObject and userObject are optional; if not specified, null must be specified in the method call.

The securityObject is a serializable object that is passed unmodified to the SecurityValidationExit, which allows customers to validate the client's security credentials. The securityObject allows customers to pass any arbitrary security-related information from the caller to the target .NET server to be evaluated by the exit.

The userObject is a serializable object that is passed unmodified to the exit UserExit. The userObject allows customers to pass any arbitrary data from the caller to the target .NET server to be evaluated by the exit.

Each exit may report an error by throwing an exception, which will be returned to the client without executing the server.

Chapter 12: Advanced Topics

This section contains the following topics:

ILDASM

ILDASM is the Microsoft MSIL Disassembler. It is delivered as part of the .NET Framework SDK. This utility is useful for examining the contents of a .NET EXE or DLL file.

To run this program, click Start, All Programs, Microsoft Visual Studio 2010, Visual Studio Tools, and Visual Studio Command Prompt (2010). At the command prompt, enter ILDASM and press the Enter key. The application starts.

You may select File, Open from the main menu to select a file to examine. Alternatively, you can drag-and-drop a file from Windows Explorer to ILDASM to open the file.

When ILDASM has opened a file, you examine the contents of the file by expanding the tree view. Double-clicking the MANIFEST opens a dialog to display several items of interest, including:

- The external assemblies and their version numbers that this .NET EXE or DLL references
- The version and public key of this .NET EXE or DLL file

Logging

There may be occasions during the troubleshooting of a problem that require additional runtime information. Logging can be performed on the client, the cooperative flow, or the server.

Important! Logging has a significant negative impact upon the performance of the generated application. This feature should only be used for short periods of time and as directed by Technical Support.

Logging CA Gen ASP.NET Clients

To perform logging on the CA Gen ASP.NET Client, locate the application's web.config file. This file is typically located in the directory named for the application under the IIS wwwroot directory (C:\Inetpub\wwwroot\appname).

In the editor of your choice, open the file. Search for the string CMIDEBUG. Change the value of CMIDEBUG from 0 (no logging) to -1 (log all possible categories). Exit the editor saving the changes.

For the CMIDEBUG setting to take effect, close the current Internet Explorer window. Open a new Internet Explorer window to access the application on a new session.

The log file, lg-<appname>-<procid>.log, (where <appname> is the application name and <procid> is the application's process id), is located in the following directory:

```
%USERPROFILE%\AppData\Local\CA\Gen xx\logs\net.
```

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Logging Cooperative Flows

To perform logging on the CA Gen cooperative flow, locate the application's commcfg.txt file. This file is typically located in the bin subdirectory under the directory named for the application under the IIS wwwroot directory (C:\Inetpub\wwwroot\appname\bin).

In the editor of your choice, open the file. Search for the string CMIDEBUG. Uncomment the statement CMIDEBUG=ON. Exit the editor saving the changes.

There are no special tasks to be performed in order for the CMIDEBUG setting to take effect. Simply perform the flow.

The log file, trace-<appname>-<procid>.out (where <appname> is the application name and <procid> is the application's process id), is located in the following directory:

```
%USERPROFILE%\AppData\Local\CA\Gen xx\logs\net
```

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Logging CA Gen .NET Servers

To perform logging on the CA Gen .NET Server, locate the application's application.config file. This file is typically located in the bin subdirectory under the directory named for the application under the IIS wwwroot directory (C:\Inetpub\wwwroot\appname\bin).

In the editor of your choice, open the file. Search for the string CMIDEBUG. Change the value of CMIDEBUG from 0 (no logging) to -1 (log all possible categories). Exit the editor saving the changes.

There are no special tasks to be performed in order for the CMIDEBUG setting to take effect. Simply perform the flow.

The log file, lg-<appname>-<procid>.log (where <appname> is the application name and <procid> is the application's process id), is located in the following directory:

```
%USERPROFILE%\AppData\Local\CA\Gen xx\logs\net
```

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Version Incompatibility Runtime Errors

The following is an example of an error where version incompatibility is called out:

```
TIRM030E: Application failed - Updates have been backed out
TIRM031E: Failing procedure exit data follows:
TIRM032E: Last or current action block id = <number>
TIRM033E: Last or current action block name = <name>
TIRM034E: Last or current database statement = <number>
TIRM035E: Current statement being processed = <number>
TIRM037E: Fatal Error was encountered ***
TIRM157E: An unexpected exception was encountered during processing
[Function: NETDynamicCoopFlow::DoFlow]BinaryFormatter Version incompatibility.
Expected Version 1.0. Received Version 1835627630.1699884645.
TIRM046E: Processing terminated ***
TIRM044E: Press OK to continue ***
```

This error, while descriptive, is both inaccurate and misleading. The real cause of the problem is that the CA Gen .NET Server being accessed does not exist.

ACTION: Ensure the target server has been installed into Component Services (COM+).

Index

O

P

R

S

T

U

V

W