CA Gen

Windows Implementation Toolset User Guide

Release 8.5

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 section contains the following topics:

Implementation Toolset

The Implementation Toolset (IT) is a collection of tools to build and run applications generated by CA Gen on a target system.

The IT includes the following tools:

- Build Tool
- Application Execution Facility (AEFN)
- CA Gen Runtime
- CA Gen User Exits

Build Tool

The Build Tool is the application that configures, compiles, and links applications generated by a CA Gen model.

Note: For more information about the build tool, see the *Build Tool User Guide*.

Application Execution Facility

The Application Execution Facility (AEFN) serves as an interface for CA Gen applications to run on a target system. It interfaces between the target platform's operating system and the application.

More information:

Testing and Running Applications (see page 43)

CA Gen Runtime

The CA Gen Runtime includes libraries the generated application calls to handle low-level and common functionality. These runtime libraries include GUI, block mode, and Web-based application support.

CA Gen User Exits

CA Gen user exits are separate, reusable code modules that are used to perform functions such as screen formatting and message switching. These user exits are part of the CA Gen runtime.

Note: For more information about user exits, see the *User Exit Reference Guide*.

Audience

This guide is intended for CA Gen developers using the Implementation Toolset on Windows platforms to build and install CA Gen generated applications.

Users should know how to configure the development platform to implement generated applications and should be familiar Microsoft Visual Studio, Oracle Database, IBM Db2, Microsoft SQL Server, CA Gen Build Tool, CA Gen Workstation Toolset, and other technologies required for the Windows platform.

Related Information

Throughout this document, the environment variable %GENxx%Gen is used. The letters xx within this environment variable refer to the current release of CA Gen. For the current release number, see the *Release Notes*.

Visual Studio Support

CA Gen supports compiling generated C applications on Windows using Visual Studio 2010. The %GENxx%Gen\VSabc folder contains a collection of files that support Visual Studio. A set of user exit rebuild procedures are also present in the %GENxx%Gen\VSabc folder and must be used to rebuild any necessary Visual Studio designated user exits. Add %GENxx%Gen\VSabc to PATH when working with C applications.

Note: VSabc refers to the supported version of Visual Studio. Replace VSabc with VS100 for Visual Studio 2010 and VS110 for Visual Studio 2012. xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

64-bit Windows Support

CA Gen supports compiling and executing generated C Blockmode and server applications as 64-bit images on Windows using Visual Studio. Users must regenerate their code to access 64-bit data types used in the Windows X 64 API to create 64-bit images.

The %GENxx%Gen\VSabc\amd64 folder contains a collection of files that support 64-bit Windows applications. A set of user exit rebuild procedures is also available in the %GENxx%Gen\VSabc\amd64 folder and must be used to rebuild any necessary 64-bit designated user exits.

If you choose to use the 64-bit runtimes provided with CA Gen to execute your application, you must modify PATH to append %GENxx%Gen\VSabc\amd64 before %GENxx%Gen\VSabc.

Note: Prepending %GENxx%Gen\VSabc\amd64 to PATH must only be done in the current command window session, and must not be set in the System Environment variables.

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*. abc refers to the supported version of Visual Studio. For more information on the supported version of Visual Studio, see the *Release Notes*.

Chapter 2: Processing Overview

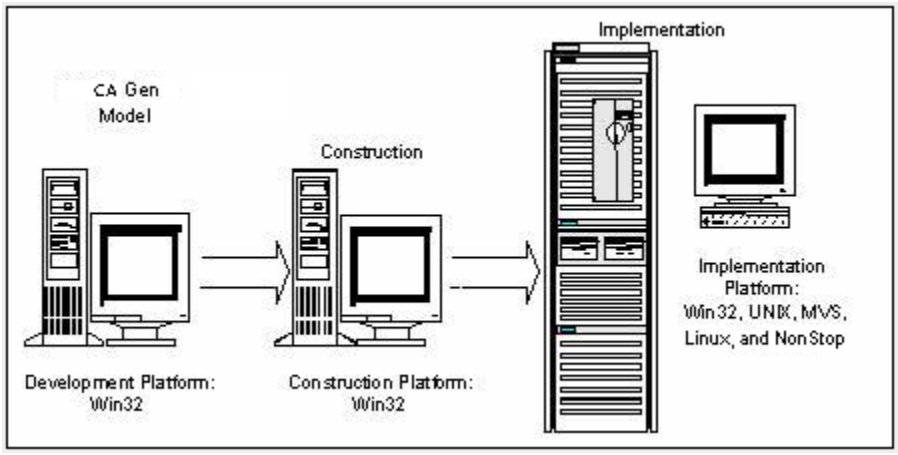
This chapter includes information about preparing a CA Gen model to use on multiple platforms.

This section contains the following topics:

Target System Implementation

Target system implementation is the process for developing an application on one platform and executing it on another platform. Using this process, you can prepare a single CA Gen model for use on many different platforms.

The following illustration describes this concept:



Applications are developed as models using various CA Gen Toolsets on a workstation called the Development Platform. Models are prepared for execution through a process called *construction*, which may occur on the development platform or on another system. The resulting components, called remote files, are transferred to a target system for compilation and execution. The remote files contain source code, data definition language (DDL), and special control information that allow the model to be installed as an application within a CA Gen environment on the target system.

The various components of the remote files define the organization and contents of the application and make it possible for CA Gen to identify and process the application on the target system. You can test and run the application on the target system within the CA Gen environment.

Each application usually includes at least the following three remote files:

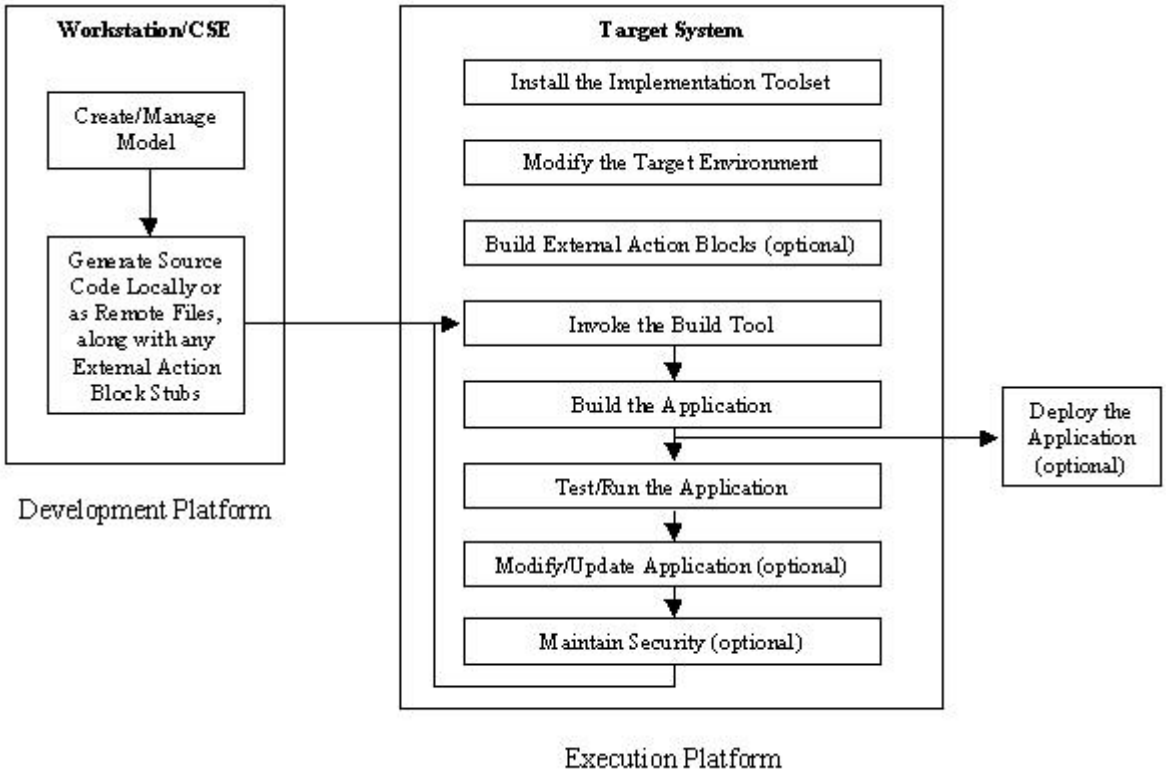
- One for the database information
- One for the referential integrity triggers
- At least one load module

When the development platform is also the target system, it is a local build. You do not need to generate the remote files. All files that would be collected into the remote files are generated and left on the local system. The Build Tool is automatically invoked to perform the build of the newly generated application.

To modify an application, update the original model on the development platform, regenerate the changed elements into remote files and move the files to the target system. In this way, you can enhance and modify the implemented systems without having to work directly with the generated code.

The following illustration shows basic workflow involved when implementing an application on a target system. The workstation tasks are performed first, then the remote files are transferred, and, finally the target system tasks are completed. The target system tasks primarily involve only the Implementation Toolset and the Setup Tool, which must be installed before the remote files can be processed.

Note: The target system may also be the development system for Windows platforms.



Note: For more information about installing the Implementation Toolset and the Setup Tool, see the *Distributed Systems Installation Guide*.

Package Remote Target Systems

You must package a CA Gen model before generating the remote files that are transferred to your target system and installed. Application models are packaged into one or more load modules. If a model is packaged into more than one load module or operations library, each load module or operations library is implemented as a separate remote file and linked on your target system into a separate executable or dynamic load library. Component Models are packaged into one or more operations libraries.

Local Target Systems

When the development system is also the target system, there are no remote files used for the application build. All generated files are immediately available to the Build Tool, and key off of the generated Install Control Modules. Although the following sections reference remote files, the same concepts are true for local builds.

Types of Remote Files

The different types of remote files are shown in the following table:

Type	Number	Description of Contents
Load Module	One or more per application	Application source code for a single load module.
Operations Library	One or more per application	Public action blocks for a single operations library.
Database	Usually one per application	DDL for an application database. For target system implementation, all load modules for a CA Gen application usually share a common database.
Referential Integrity	One per application	RI trigger routines for an application. These routines implement referential integrity for the entire application.

Install Control Module

Every remote file has an Install Control Module (ICM), or install deck, that identifies and provides instructions for installing all components of the remote file. The ICM is in Generalized Markup Language (GML) format.

The ICM identifies each component of a load module, database, or RI trigger set, so that it can be properly installed on your target system. The contents of the ICM depend on the packaging selected. The ICM carries information for the complete load module, database, or RI trigger set, even if code is generated only for one portion of the specified remote.

Load Module Remote Files

The load module remote file includes generated code in a high-level language. This code contains a Dialog Manager, one or more procedure steps, screens, and the action blocks used by the procedure steps. The Dialog Manager is created by CA Gen, based on the control flow of the procedure steps and action blocks you have packaged into this load module. You specify the procedure steps and action blocks contained in each load module during packaging.

Dialog Manager

During remote file generation, CA Gen builds a Dialog Manager for every load module. The Dialog Manager controls the dialog flow (link and transfer) between procedure steps, supports terminal input/output, and maintains the execution context in a profile table.

The Dialog Manager always receives control from the TP monitor (usually the AEFN) when the load module is executed. The Dialog Manager determines which procedure step within the load module to execute. The input view of the procedure step is then populated from screen input, data passed from another procedure step, and data taken from the profile source. This allows each procedure step to reference data in its input view without considering how the data values are obtained.

The Profile Manager is the part of the Dialog Manager that manages the profile table. The profile table is a temporary runtime stack that stores the export view of the procedure steps. It is used to pass information between procedure steps.

Operations Library Remote Files

Used in packaging Component Models, these files contain a collection of public operations that result in a dynamic load library. Operations library remote files only contain action blocks.

Database Remote Files

The database remote file contains the DDL statements and installation information. You may choose to generate DDL for only a part of the database. This feature is appropriate when a database has already been installed and some changes are made to the data structure diagram.

Referential Integrity Trigger Remote Files

The Referential Integrity (RI) trigger remote file contains the code for implementing referential integrity. When a record or field is deleted from a database, referential integrity ensures that all other records or fields that depend upon the deleted record or field for their identity are also deleted. Each model has only one RI trigger remote file.

The name for RI trigger remote files defaults to CASCADE. You can change the name from CASCADE to a different name on the member name of the RI trigger before construction in the Workstation Toolset.

Note: The RI trigger remote file contains the trigger routines for the entire data model residing on the workstation when generation occurs. If the RI trigger remote file is generated from a subset of the complete data model rather than from the complete data model, you may get an incomplete set of trigger routines.

The RI trigger routines are compiled and placed in a library. They are available to be linked into individual load modules as needed.

Transfer Remote Files to the Target System

You choose the transfer process to move remote files from the development platform to the target system. You may use the Build Tool to transfer files.

Note: For more information, see the *Build Tool User Guide*.

External Action Blocks

To access information not directly available to a generated application, you use an external action block. For example, a load module may need to use a standard date manipulation subroutine that has been defined for a particular organization or the load module may need to access files that were not created with CA Gen.

An external action block defines the interface between the CA Gen procedure step or action block that invokes it and the logic created outside of CA Gen, so that information can be successfully passed back and forth. This structure is used to match the views of a procedure step with the views (input and output arguments) of a handwritten subroutine.

When you use a CA Gen model to generate remote files for installation on a target system, an external action block stub is created that provides the source language framework for the external action. This stub includes the following information that CA Gen can supply from the definition of the external action block:

- Input data definitions
- Output data definitions
- Parameters
- Framework for the actual code

The only thing missing is the action logic.

During the load module packaging portion of construction, include external action blocks in the load modules just as you include other CA Gen procedure steps and action blocks. When CA Gen generates that load module, it includes information identifying the action block in the ICM for that remote file. That remote file does not include the action block stub. You must move it to your target system separately.

For your application to execute properly on your target system, you must add the appropriate logic to the generated action block stub before you compile and install it. A number of steps are required before the load module containing the external action block is built and implemented. These steps are listed and explained later in this guide.

CA Gen Runtime User Exits

Certain system functions, such as retrieving a user ID or handling errors, may vary in implementation from one target system to another target system because of the combination of hardware and software being used in that target system.

Runtime user exits are standard routines that allow all generated applications to access these system features. Runtime user exits reside on the target system and can thus be accessed by each application without actually being coded as part of it. These routines can be used as they are to supply basic functionality, or they can be customized to the needs of your particular target system.

Chapter 3: Prerequisite Implementation Tasks

The following list provides several sets of tasks that are required before building and executing your application:

- Pre-generation tasks
  - Packaging EABs
  - Considerations for Remote generation
  - Considerations for Trace generation
- Pre-build tasks on the target system
  - Installing the DBMS
  - Considerations for user access
  - Installing the Implementation Toolset
  - Customizing delivered files

This section contains the following topics:

Prerequisite Tasks on the Development Platform

To ensure that your target implementation is successful, you must perform the following pre-generation tasks and define specific parameters on the development platform before generating your remote file:

- Packaging EABs
- Considerations for Remote Generation
- Considerations for Trace Generation
- Considerations for Performance

These tasks and parameters are explained in the following sections.

Package External Action Blocks

During the load module packaging portion of construction, you can include external action blocks in your load modules in the same manner as other CA Gen procedure steps and action blocks. When generation occurs for that load module, information identifying the action block is included in the ICM for that remote file. However, the action block stub is not included in the resulting remote file. It must be moved to your target system separately. Similarly, if you hand-edit code into the external stubs, you should move the stubs out of the source code component subdirectory so that subsequent generations do not overwrite the edited stubs.

For example, on the Windows code generation platform, the external action block code must be moved from the source code component subdirectory (\c for the C language) associated with the model being implemented. The file containing external action block code is named using the action block name and a file extension appropriate to the language being used (.C).

Remote Generation Considerations

The following sections explain the prerequisites you must consider for remote generation.

Target System

During construction, ensure that you specify the correct operating system, database management system (DBMS), programming language, TP monitor, and communications that will be used for implementation.

Install Control Modules

You can generate code for all the components that are packaged into the load module, or select specific components for generation. Regardless of the selection of components to generate, the installation must be performed on the entire load module. The installation has some important implications:

- Each time remote or local installation is requested for load module code generation, an ICM is created for that load module.
- After the ICM and all specified load module components are generated, the ICM and all load module source components found in the language subdirectory (\c for the C language) for the model are copied into a single remote file.
- You need to consider subset definitions carefully for the purpose of code generation.
  - If the load module is generated from a subset of a CA Gen model, the ICM generated for the load module is based on the packaging information visible in the subset used for generation.
  - If your subset does not contain all the components of a load module, the ICM may be incomplete. If this load module is installed on your target system, the incomplete ICM will overwrite any previous ICM definition. Only the components shown in the new ICM will be installed when the load module is rebuilt. An incomplete ICM could cause the application to fail.
- All components included in a load module definition are included in the generated remote file as long as the components are stored in the directory where CA Gen can find them, even if only one component is updated.
- The complete load module is defined in the ICM regardless of the number of components you have modified. (This is done for maintenance purposes. It lets you change and regenerate portions of a load module without changing its definition.)
- An incomplete load module ICM can result if the subset used to generate the load module does not contain all the elements for that load module.

Database DDL Files

During Construction on the workstation, do not qualify tables and indexes with your owner ID. Specifying your owner ID is useful only when testing on your workstation and may present problems for another user attempting to install the DDL.

Because CA Gen manipulates data according to the type of targeted DBMS, existing data might not be supported by CA Gen. A field that is logically used to store time-of-day data in Oracle is really stored in a DATE column with year, month, day, hour, minute, and second information. CA Gen will default an unspecified year/month/day to 01/01/0001 (January 1, 0001) before sending it to Oracle. This may be different for other DBMSs.

Load Module Remote Files

When generating load module remote files, do not delete the source after installation completes. Doing so may indirectly cause the building of load modules to fail.

When more than one load module uses the same common action block, the first load module to be generated contains the actual code. When you delete the source after installation, the source code is deleted from the code generation platform after the first module is formatted into a remote file. All other remote files contain only the reference to this action block. If the remote files are built in a different order than the order in which they were created, a load module that references the action block will be built before the load module that actually contains the action block. In this case, the build will fail.

Operations Library Remote Files

The operations library remote file contains ICM information and all the action block modules defined in component packaging. This remote file is a subset of functions that will be built into a library and used to build a load module.

Note: If a load module uses one or more operations libraries, each operations library and the load module must use the same DBMS or use no DBMS.

RI Trigger Remote Files

The RI trigger remote file contains the trigger routines for the entire data model residing on the workstation when generation occurs. If the RI trigger remote file is generated from a subset of the complete data model rather than from the complete data model, you could get an incomplete set of trigger routines.

Trace Generation Considerations

The trace function can be used only if you have generated code with trace support. The resulting source code contains all the additional logic necessary to allow the special trace functions to occur.

You can generate trace for a single procedure step, an entire load module, or your complete application.

The special code included to allow trace significantly increases the size of each generated source module. If space is a potential problem on the code generation platform or your target system, add trace selectively rather than globally.

You can selectively test portions of your application in the following ways:

- You can generate the trace code only for the elements you want to test during load module generation.
- You can elect to turn trace off during runtime if you have generated your entire load module or application with trace,

Performance Considerations

Ensure that you specify dialog flow definitions and packaging options independently. An application runs efficiently if each load module contains procedure steps that are likely to be used together. Therefore, coordinate the packaging with the dialog flow definition when packaging your application,

Note: Applications generated with trace support run slower than the applications generated without trace support.

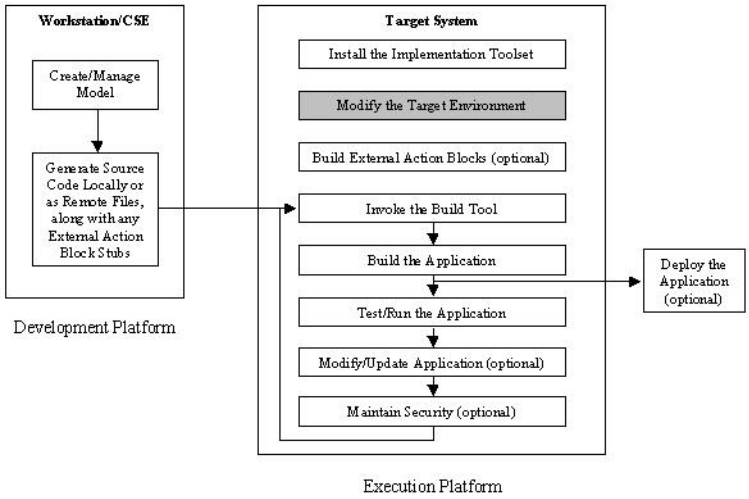
Prerequisites on the Target System

You must perform the following tasks to successfully implement a generated application on your target system:

- Installing the DBMS
- Considerations for user access
- Installing the Implementation Toolset
- Customizing delivered files

These tasks are explained in the following sections.

The following illustration describes the relationship of modifying the target environment to the other required and optional IT tasks:



Install the DBMS

For information about environment settings that must be set before use, see the DBMS software documentation.

User Access Considerations

To successfully implement an application on a target system, you must have the appropriate levels of access to the volumes, subvolumes, and files being used by the Implementation Toolset.

Install the Implementation Toolset

Note: For information about this task, see the *Distributed Systems Installation Guide*.

Select and Customize Scripts

A script controls the processing of each load module, RI trigger set, operations library, and database DDL .The Build Tool includes a set of scripts you can customize to include many types of information specific to your target system.

Note: For more information about selecting and customizing scripts, see the *Build Tool User Guide*.

Define Environment Variables

The Implementation Toolset uses environment variables to specify paths and other configuration information necessary to install and use CA Gen applications. The environment variables set up macros, or aliases, referenced in several places, including MAKE files, source code, and script files. Each user's .login file must set the variables to ensure they are properly set during system logon, before invoking the Build Tool or the AEFN.

To display the current environment, use the set command. To view the current value of an environment variable, use the echo command as shown in the following example:

```
echo %AEPATH%
```

Required Variables

The following table describes the set of environment variables the Implementation Toolset requires.

Name	Description
AEPATH	Search path for executable load modules. A set of semicolon-delimited directory paths indicate the search order for the user's aeenv file and inqload directory, allowing the Build Tool and AEFN to reference multiple directories for test and production applications in specific order. Ensure that %GEN*xx*%Gen*yy* is in AEPATH. You must use the following format: AEPATH=[directory_1;directory_2;...directory_n;%GEN*xx*%Gen*yy*] **Note:** *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*. **Note:** *yy* refers to the compiler version folder which will be used when executing the Implementation Toolset. For example, VS100\amd64.
IEFGXTP	This environment variable represents the directory that contains the codepage translation files. IEFGXTP=[directory]
IEFH	CA Gen IT home directory. The CA Gen software installation directory used to locate the runtime user exits, reusable code packages, when linking a generated application. Before starting an CA Gen applications, you must set this variable using the following format: IEFH=[directory]

Name	Description
PATH	Search path for executables. A set of directory paths indicating executable program search order. The path must include %IEFH% and DBMS bin directories, such as %DB2DIR%\bin or %ORACLE_HOME%\bin. You must use the following format: PATH=[directory_1;directory_2;...directory_n]
PTHOME	Specifies the directory where Profile Table files are created, or where previous Profile Table files are located. You must use the following format: PTHOME=[directory]
PTOPT	Specifies to use the Profile Table. You must use the following format: PTOPT=[YES or YY] ■ **YES**—Use Profile Table files ■ **YY**—Compress PT Files

DB2-specific Variables

The following table explains the DB2-specific environment variables.

Name	Description
DB2DIR	Name of DB2 home directory, where DB2 is installed
DB2INSTANCE	Name of DB2 database Instance
INSTHOME	Location of DB2 database Instance
DB2DBDFT	DB2 Database default instance name

Oracle-Specific Variables

The following table explains the Oracle-specific environment variables:

Name	Description
ORACLE_HOME	Oracle Home directory
ORACLE_SID	Oracle System ID that specifies the Oracle database to use

Configuration Files

The IT includes many configuration files that you can modify after installing the IT, including build script files, initialization files, sample files, and user exit files. If you reinstall the IT, the installation preserves your changes to these files.

Note: You must rebuild binary files, such as dynamic link libraries or images built from modified configuration files after reinstalling.

The CA Gen 8.5 IT includes the following configuration files, residing in the %GENxx%Gen directory:

- p3270key
- codepage.ini
- application.ini
- tirdb2.ppc
- tirdconn.sqc
- tircodbc.c
- tirodbc.c
- tirora.pc
- tiroconn.pc
- tirdcryp.c
- tirdlct.c
- tirdrtl.c
- tirelog.c
- tirhelp.c
- tirmtqb.c
- tirncryp.c
- tirsecr.c
- tirsecv.c
- tirserrx.c
- tirsysid.c
- tirterma.c
- tirupdb.c
- tiruppr.c
- tirurtl.c

- tirusrid.c
- tirxlat.c
- tiryyx.c
- cictuxwsx.c
- cictuxx.c
- cieciclx.c
- cimqclex.c
- cimqsvex.c
- citcpclx.c
- ciwsclx.c
- csuglvn.cxx
- proxyxit.c
- bt\scripts\build_ddl.scr
- bt\scripts\build_lm_c.scr
- bt\scripts\build_ri_c.scr
- vs100\aeenv
- vs100\amd64\aeenv
- vs110\aeenv
- vs110\amd64\aeenv

Build User Exits

The user exits are in C to support applications generated for C. You can modify the user exits delivered with the CA Gen runtime. You must modify the user exits before building the application.

Note: For more information about the available user exits and how to rebuild them, see User Exits (see page 65).

Chapter 4: External Action Blocks

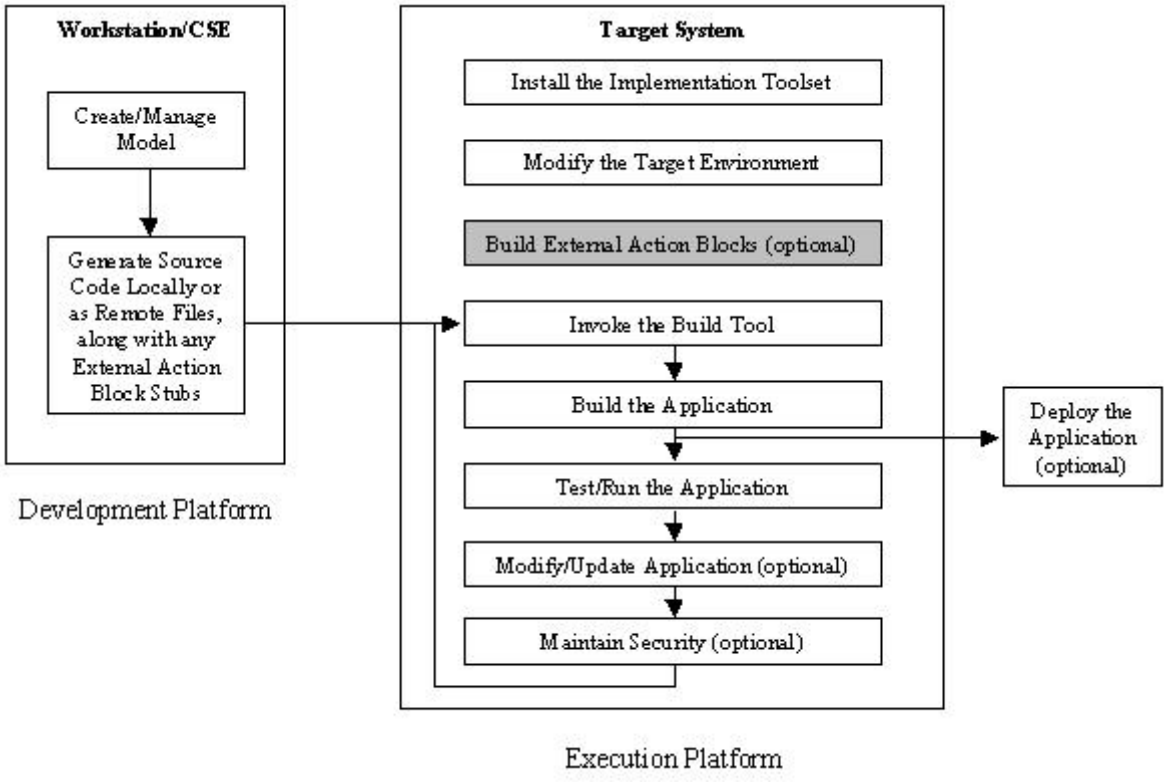
This section contains the following topics:

External Action Blocks

An External Action Block (EAB) defines the interface between the procedure step or action block that invokes it and the logic created outside of CA Gen (that is, handwritten subroutines). An EAB provides a structure that is used to match the views of a procedure step with the views (input and output arguments) of a handwritten subroutine.

Note: For a list of conditions that must be satisfied on the development platform prior to implementing EABs on your target system, see Package External Action Blocks (see page 22).

The relationship of building EABs to the other required and optional IT tasks is described in the following illustration:



The single action statement EXTERNAL, which appears in the action block itself, distinguishes EABs from action blocks that are generated by the current model.

EABs are created when code is generated. The CA Gen software cannot generate actual code for EABs, but it does create a stub during code generation. This stub specifies the information that your external subroutine provides to the CA Gen application through the EAB and the information that your external subroutine expects to receive.

When an EAB is generated, the CA Gen software creates:

- The subroutine source code required to allow the EAB to interface successfully with other generated action diagrams. This includes coordination of input and output data and other parameters.
- A stub, or comment, indicating where subroutine source code is required to complete the logic of the EAB.

To make the action block usable, add either a call to an existing subroutine or the logic necessary for the action block to perform its function.

For an application to properly execute on a target system, the EAB file is copied to the target system where the appropriate logic must be added to the generated action block stub before it is compiled and installed. A number of steps are required before the load module containing the EAB is built and implemented. These steps are outlined in the next section.

How EABs are Created

EABs must be completed and installed on your target system before the load modules that invoke them are built.

Note: Completing EABs requires a significant amount of programming experience and a detailed knowledge of the programs or the information being accessed using the EAB.

Each of the following tasks is performed outside the CA Gen software. Before you build a load module that uses an EAB you need to perform each of the following procedures:

Follow these steps:

1. Identify any EABs that need to be completed.
2. Locate the EAB code within your target configuration.
3. Analyze the EAB stub to determine the input and output requirements for your external action.
4. Create the appropriate external action logic.
5. Compile the EAB.
6. Link into an EAB library.

Identifying External Action Blocks

The Install Control Module (ICM), associated with each load module contains an EAB identification section. The section begins with the Generalized Markup Language (GML) tag :extern and ends with :eextern. Between the beginning and ending tags are the names of all EABs referenced by the load module.

There is an EAB identification section for each EAB referenced by the load module. All EABs in the load module are identified in the same section of the ICM.

If the ICM for a load module you are working with contains :extern and :eextern tag pairs, you have EABs that need to be completed before building this load module.

The following example identifies two EABs from an ICM:

```
:extern
member=EXTERNAL1
name=XTRN_SAM1
techsys=BUS_SYSTEM
:eextern.

:extern
member=EXTERN2
name=XTRN_SAM2
techsys=BUS_SYSTEM
:eextern.
```

Locate External Action Block Code

During the load module packaging portion of construction, EABs are included in load modules just as other procedure steps and action blocks are included. When generation occurs for that load module, information identifying the action block is included in the generated ICM. However, the action block stub is not included in the resulting remote file. It must be moved to your target system as a separate file.

On the development platform, the EAB code must be moved from the source code component subdirectory (\c) associated with the model being implemented to the target system. The file containing EAB code is named using the action block name and a file extension appropriate to the language being used (.C).

When you move the action block to the target system, store it in any directory until the code is compiled. After compile, store the EABs in the file to which LOC.EXTERNAL_LIB Build Tool script token points. This is most likely a shared or archived library. The file LOC.EXTERNAL_LIB identifies is used during the load module link to add the EABs to the load module executable.

Note: For more information about the LOC.EXTERNAL_LIB token, see the *Build* Tool *User Guide*.

Analyze External Action Blocks

CA Gen uses the views defined in the EAB to generate the stub for the interface routine. From the views in the EAB, determine:

- The data that the EABs will receive from the generated application
- The data that must be returned to the generated application
- Any processing that will be required to implement the EAB

The import and export views shown in the stub are supplied by the procedure step or action block that invokes the EAB. The import views define the data that is passed from the generated application to the interface routine. The export views define the data returned to the generated application from the interface routine.

Decimal Precision Attributes

Import and Export views in External Action Blocks may contain any of the supported attributes by CA Gen. This section is intended to give you a better understanding for handling views that contain attributes of type decimal precision.

The C language data structure for an attribute that is implemented with decimal precision is a DPrec array whose size is the length of the attribute plus 3 (for the sign, decimal point, and null terminator). A DPrec is a typedef of char.

Example:

An attribute that is defined as a number of 18 digits will be implemented as DPrec[21].

The number represented within the DPrec array may consist of the following characters depending on the definition of the attribute it implements:

- A minus sign (-)
- Zero or more decimal digits with a decimal point
- One or more decimal digits without a decimal point
- A decimal point
- One or more decimal digits
- A null terminator

For the import view, a decimal precision attribute may be its simplest form or may contain a plus sign and leading with trailing zeros.

For the export view, all decimal precision attributes should adhere to the following rules:

- The character representation of the number placed in the DPrec array may contain fewer digits than defined for the attribute. However, it must not contain more digits to the left or right of the decimal than defined for the attribute.
- The DPrec array must be null terminated.
- If number of digits to the right of the decimal equals the total number of digits, the resulting string must not contain a leading zero before the decimal point.

Create External Action Logic

You can create logic for an EAB in the following ways:

- Add a call to an existing subroutine.
- Write code for a new subroutine, and add a call to that subroutine in the EAB.
- Add logic directly into a generated stub.

  When you use the generated stub, much of the work is already done in a format that is acceptable to the CA Gen software.

You can write EAB code in any language. Specific requirements exist, however, for the action block name and the order of parameters passed to and from the generated load module.

The generated load module passes the following parameters to the EAB interface routine. The parameters are passed in the following order:

For C with High Performance View Passing

Follow these steps:

1. IEF-RUNTIME-PARM1
2. IEF-RUNTIME-PARM2
3. PSMGR-EAB-DATA (null array)
4. w_ia (import view C)
5. w_oa (export view C)

Note: For more information about High Performance View Passing, see the Host Encyclopedia Construction User Guide or the Toolset Online Help.

For C Without High Performance View Passing

Follow these steps:

1. IEF-RUNTIME-PARM1
2. IEF-RUNTIME-PARM2
3. w_ia (import view C)
4. w_oa (export view C)
5. PSMGR-EAB-DATA (null array)

IEF-RUNTIME-PARM1 and IEF-RUNTIME-PARM2

Identifies the first two parameters passed from the generated load modules. These parameters must be coded on the entry statement of the interface routine and must always be passed in this order to any subordinate routines that are called.

w_ia and w_oa

Identifies the position of the import and export views () for C language depending on whether High Performance View Passing is used (see the previous table).

Note: High Performance View Passing impacts the order of parameters transferred into an EAB. By default, High Performance View Passing is set on.

PSMGR-EAB-DATA

Identifies an array set to zeroes (null array). For target system implementation, this array is passed but not used (the array is used for IMS and CICS applications on the mainframe).

Note: For more information about the use of PSMGR-EAB-DATA, see the *Host Encyclopedia Construction User Guide*.

Note: For component development, High Performance View Passing must be set on for both the component and the consuming model.

The interface routine must contain data structures that correspond exactly to the import and export views of the EAB. The fields in the data structures correspond to attributes in the import and export views of the EAB.

Note: Each attribute field in the data structures must be preceded by a one-byte field defined in C as char. This one-byte field contains a value that must not be changed.

Modify the stub using any editor that can save files in the appropriate character set format. The modified stub cannot contain any control codes or header information unique to the editor.

Compile, Link, and Store an External Action Block

After completing the EAB, you must:

- Compile the EAB (and any newly-written subroutines called by the stub).
- Link the EAB into an object library.
- Define the fully qualified file name for the object library through the Location Details screen of the Setup Tool.

Note: For instructions on how to compile and link your EAB on your particular target system, contact your development organization.

After you complete all the required activities, you are ready to build load modules that use the EAB.

Recompile and Relink an External Action Block

When migrating to the latest CA Gen environment, it may be necessary to recompile and relink your External Action Blocks. This may be due to a number of reasons, including changes to the calling interface, compiler upgrades, and third party product upgrades. If you recompile or relink your EAB, you will need to relink your application to include the rebuilt EAB.

Note: For release specific instructions, see the *Release Notes*.

Testing an External Action Block

After the load module that uses the EAB builds successfully, you are ready to test the load module. For more information, see the chapter "Testing and Running Applications".

More information:

Testing and Running Applications (see page 43)

External Action Blocks in Component Modeling

When an EAB is packaged into an operations library, you must compile the EAB separately and store it in an externals library before you can build the operations library. Set the Build Tool script token LOC.EXTERNAL_LIB to the name of the externals library before building the operations library.

If the EAB is a public operation in the operations library, the consuming application's listing file must specify the lib file for the externals library.

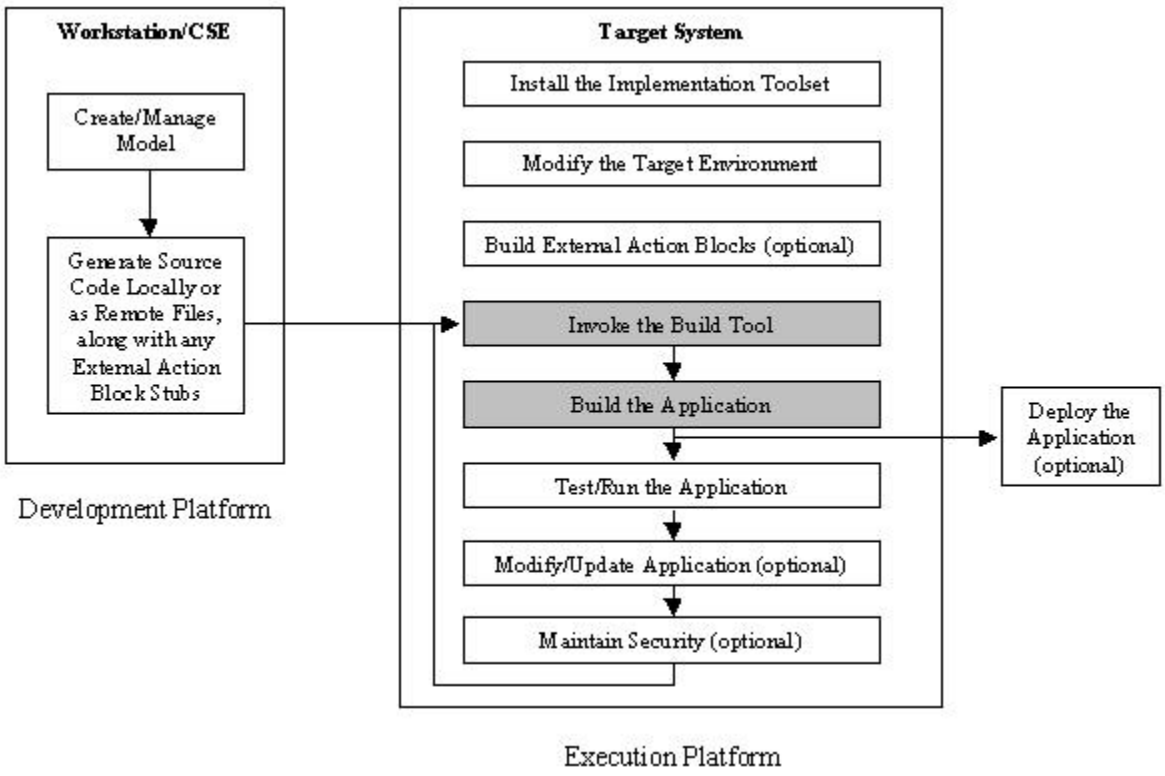
If the EAB is a private operation in the operations library, the user must ensure that no function in the consuming model uses the same name as the private EAB function. The Build Tool cannot guarantee the outcome of an application that has multiple declarations of the same function.

Chapter 5: Processing Generated Files

Before processing generated files, you must:

- Set up the target environment
- Rebuild the user exits
- Compile the External Action Blocks (EABs) and put them in the correct directories

The following illustration shows the relationship between processing generated files and other IT tasks:



This section contains the following topics:

Invoke the Build Tool

After generating the load modules, RI triggers, database DDLs, and operations libraries on the development platform, they are:

- Transferred to the target system, through the Build Tool or outside of CA Gen, when building remote files on a system other than the system that generated the code
- Processed by the Build Tool into an executable application to test and run

Note: For more information about invoking and using the Build Tool, see the *Build Tool User Guide*.

Operations Library Considerations

You must generate and build an application with one or more operations libraries in a specific sequence.

If the application has operations that include common action blocks, you must:

- Generate and build the operations library you generated
- Build the load modules that reference the operations library

If the application references an operations library built from another model, you must build and specify the operations library in the Build Tool before generating and building the load modules that reference it. The Build Tool profile entry set is OPT.CBDLIST under Options.

Note: For more information about setting tokens, see the *Build Tool User Guide*.

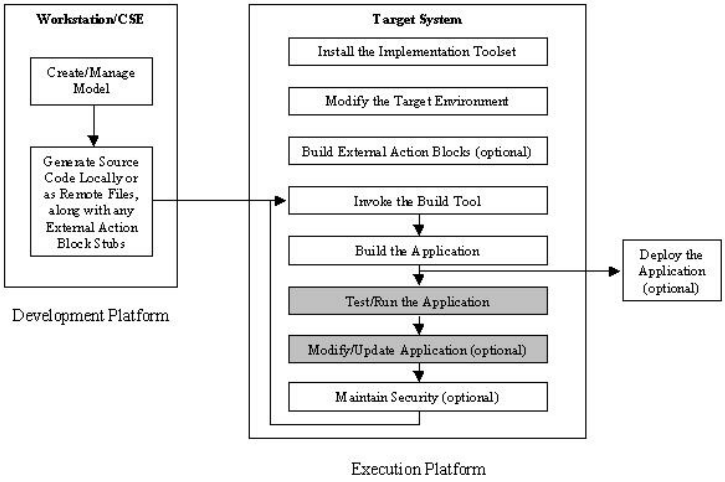
Recompile and Relink Application Considerations

When migrating to the latest CA Gen environment, it may be necessary to recompile and relink your application. This may be due to a number of reasons, including operating system upgrades, compiler upgrades, and third party product upgrades.

Note: For release specific instructions, see the *Release Notes*.

Chapter 6: Testing and Running Applications

Although an application is ready for execution after processing the generated files, we recommend testing the application using the Diagram Trace Utility before introducing it into the production environment. The following illustration shows the relationship of testing and running the application to the other IT tasks.



This section contains the following topics:

AEENV File

CA Gen creates a special file to hold the transaction codes associated with executable load modules. This file is named AEENV and a copy of it resides in the models' source directory where you built your application. All load modules that have been built as part of a block mode application or a cooperative server write their transaction codes to this file.

Each transaction code should be unique within each AEENV file. Duplicate transaction codes can cause problems during load module execution.

You can edit the AEENV file with any editor that saves files in ASCII format. The edited file cannot contain any control codes or header information unique to the editor.

Add transaction codes to the AEENV file in the following format:

```
tran_type tran_code lm_name user_id password database
```

where:

tran_type

Transaction Type:

- TRAN—used to invoke a load module with block mode applications
- FLOW—used to flow between load modules with cooperative servers

tran_code

The transaction code used as the key to the load module to invoke.

lm_name

The name of the executable load module file associated with the transaction code. Omit file suffix.

user_id

A user ID to access the database.

password

A password to access the database.

database

The name of the database bound to the load module.

Separate parameters with spaces and omit tabs. You must include the following parameters: tran_type, tran_code and lm_name.

You only need to specify a database when the executable load module is bound to a database. You only need a user ID and password when database access requires a user ID and password.

Block Mode and Server Executable Locations

When a C block mode interactive application, one that requires using the AEFN as the interface, or a set of C Servers are built, the executables and the triggers library are copied to the \inqload subdirectory, and a local version of the AEENV file is updated within the models' source directory, matching the behavior on UNIX and Linux systems, and avoiding the trigger library collisions that occurred when a pre-existing trigger library was in the %IEFH%\inqload directory. Previously, because the file was *not* being overridden, it was out of sync with the executables.

When a C block mode command line application is built, the executables and the triggers library remain in the models' source directory and a local version of the AEENV file is updated in the models' source directory.

Separating the locations of the executables allows both block mode application styles to co-exist for one model.

To successfully invoke these executables and servers, you must properly set the AEPATH environment variable.

Application Startup Parameters for Block Mode

For block mode applications, startup parameters supply information to the executable procedure without entering the information on the screen. The executable procedure must be an online procedure that accepts clear screen input. Use the Dialog Flow Diagramming Tool to define clear screen input.

The unformatted input is a transaction code followed by one or more parameters. Separate the parameters with the appropriate string and parameter delimiters. Use the Business System Defaults to define the string and parameter delimiters.

Note: For more information about clear screen input, see the *Block Mode Design Guide*.

Use keywords to define the parameter list. Keywords are labels that identify the parameter and must be accompanied by the parameter data enclosed in delimiters.

The format for parameters is:

```
tran_code KEYWORD1 data1, KEYWORD2 data2 . . .
```

where:

tran_code

The transaction code to invoke in the load module.

<space>

The string separator.

KEYWORD1, KEYWORD2

Keywords

<space>

Keyword or data separator.

data1, data2

Data associated with the keyword used by the executable procedure..

<comma>

Parameter delimiter.

If spaces are not selected as the parameter delimiter, they are ignored. The keyword can be the prompt that associates the field on the screen to the procedure step, or a label that associates the field to the procedure step. Keywords must be unique.

When using keywords, the parameters can be in a different order than the parameter list on the screen. For example, the following clear screen inputs are equally acceptable:

```
tran_code NAME Mary Valdez, NUMBER 123-45-6789
```

```
tran_code NUMBER 123-45-6789, NAME Mary Valdez
```

Keywords are case-sensitive and cannot contain delimiters. You can append symbols. For example, you can append an equal sign (=) to the keyword NAME, and it would appear as NAME=Mary Valdez.

If you do not define parameters with a keyword, you must enter the parameters in the same order as they are defined in the list on the Dialog Flow Diagram.

Two parameters predefined to CA Gen procedures are RESET and RESTART. These parameters apply only to online procedures of block mode applications. You must specify in the Environment Tool if you want your application to also support the RESTART parameter.

The RESTART parameter lets you redisplay an interrupted dialog. RESTART restores all data to the screen that was present when the interruption occurred. RESET allows you to redisplay a screen by starting the dialog at the beginning. Data from the interrupted dialog is not restored to the screen.

Note: For more information about RESET and RESTART, see the *Toolset Help*.

Application Startup Switches for a GUI

For GUI applications, startup switches invoke the application. They do not pass data to an executable procedure. GUI applications do not support clear screen input. You cannot supply data for an application in the startup switch entry field for GUI applications during testing from the Build Tool.

Note: The RESTART and RESET parameters do not apply to GUI applications.

The following is a list of the switches that can be provided when invoking a GUI application:

Initial PStepCode

The initial transaction code to use. When omitted, the GUI uses the transaction code associated with the primary window.

Initial command

An optional starting command that causes the procedure to execute first instead of displaying.

/db=

The name of the local database.

/dbname=

The name of the local database (same as /db=).

Note: To override the default database while testing a GUI application, specify the database name using the /db= parameter when invoking the application.

Global Database Information Container for GUI

The Global Database Information Container (GDIC) is a dynamic-link library to gather and supply database connection information for CA Gen generated GUI applications. It implements a shared cache of connection profile entries, or nodes, that GUI applications access for connection information.

The file iefgdic.ini stores information for these nodes. The information includes the following keyword sets:

Keyword	Value
Database	Database to access. dbname stores this database name. Database is the key to use in the iefgdic.ini lookup.
Trancode	Transaction code in a load module to connect from.
LoadModule	Specific load module to connect from.
Server	Database server name. Required for connection. (ODBC only)
SQLid	Database connection identifier. Required for connection.
SQLpassword	Database connection password. Required for connection.

There are one or more keyword sets in the [*RDBMS*] section, where *RDBMS* is a supported GUI RDBMS: Oracle, DB/2 or ODBC. Each keyword in the keyword set is appended with an incremental value to help identify each keyword set.

Example:

```
[MaxDatabases]
Max=100
[ORACLE]
DatabaseCount=2
Database1=ABLK05
Trancode1=
LoadModule1=
Server1=
SQLid1=ptauto
SQLPassword1=ptauto
Database2=ora102
Trancode2=
LoadModule2=
Server2=
SQLid2=ptauto
SQLPassword2=
[DB2/2]
DatabaseCount=1
Database1=ABLK05
```

```
Trancode1=
LoadModule1=
Server1=
SQLid1=ptauto
SQLPassword1=ptauto
[ODBC]
DatabaseCount=1
Database1=ABLK05
Trancode1=
LoadModule1=
Server1=ABLK05
SQLid1=ptauto
SQLPassword1=ptauto
```

In this example, MaxDatabases has value of 100, indicating this file can hold up to 100 keyword sets. There are three (3) RDBMS sections: Oracle, DB/2 and ODBC. The Oracle section has two (2) keyword sets, and DB/2 and ODBC only have one (1), identified by DatabaseCount.

The iefgdic.ini file stores the database connection information and can be in multiple locations. The system searches for the file in this order:

- %IEFGDIC% - only used for the iefgdic.ini file
- %USERPROFILE%\AppData\Local\CA\Gen xx\cfg\client
- %ALLUSERSPROFILE%\CA\Gen xx\cfg\client

If the system fails to locate the iefgdic.ini file using this search order, it creates it in the %USERPROFILE%\AppData\Local\CA\Gen xx\cfg\client directory.

If a database DLL requires connection information, it calls a GDIC function to get the information from the iefgdic.ini file for the calling DLL. The GDIC displays an input window, pre-populating as many fields as possible, and allows the user to fill in the blanks or overwrite the pre-populated fields. If all the required fields are extracted from the iefgdic.ini file, the GDIC bypasses the input window. If you look back at the previous example, the first keyword set for each RDBMS has all the required fields, allowing the GDIC to bypass the input window for the specified database.

Variables

Use environment variables to specify connection information when the keyword set for a node does not have default values for Server, SQLid, or SQLPassword.

These environment variables are:

- dsquery—Key name for Server
- dsuser—Key name for SQLid

- dspswd—Key name for SQLPassword

Note: When using ODBC, substitute dsserver for dsquery.

These environment variables complement the retrieved keyword values in the iefgdic.ini file. If the combination of the retrieved keyword values and the environment variables provide all of the required values to connect to the RDBMS, the GDIC bypasses the input window.

GUIEnvironmentVariables.ini

The file GUIEnvironmentVariables.ini file contains a set of environment variables that modify default GUI behavior and that the GUI client reads and sets. The file contains a set of commented environment variables. To use these environment variables, remove the comment from the variable (";") and modify the value.

The GUIEnvironmentVariables.ini file contains the following environment variables:

Environment Variable	Description	Default
IEFCOMPATIBLEAPPEARANCE	The 3D Window Border look is the default for all Primary Windows. To suppress the 3D look, set this variable to any value. The default 3D look makes the borders thicker.	3D Window Border look
IEFDONTFORCESCROLLBAR	If a list box has scroll events defined, the list box includes a vertical scroll bar even when it is not full. Setting IEFDONTFORCESCROLLBAR shows the scrollbar only when the list box is full.	Always show vertical scrollbar in list box
GROUPCONTROLS	When defined, the controls in a group box, not a radio button group, are moved or hidden together with the group they belong to. When undefined, only the group is moved or hidden.	Only group box is moved or hidden

Environment Variable	Description	Default
IEFTRACEGUIOBJETS	When set to Warning, display error message boxes when GUI Automation objects are not released.	No Messages displayed
IEFFLEXEDITING	Allows automatic completion of time inputs, with zeros.	No automatic completion
USEOLDSHELLCOLORS	When defined, uses the system Window color, usually white, as a window's default background, instead of the system 3D object color, usually gray.	Gray window background
IEFLOCALVIEWALLOCSCOPE	When set to GLOBAL, shares uninitialized local views between distinct instances of an action block.	Local views are not shared
RETRY_LIMIT	Determines the number of times an application tries to connect to the database.	10
USERHANDLEQUOTAPERCENTAGEUSED	Displays a warning message when the application has used this percentage of the max allowed User Handles and asks the user to close application windows. The HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\USERProcessHandleQuota registry entry lists the maximum number of handles.	90

The system searches for the GUIEnvironmentVariable.ini file in this order:

- The Load Module directory
- %USERPROFILE%\AppData\Local\CA\Gen xx\cfg\client
- %ALLUSERSPROFILE%\CA\Gen xx\cfg\client
- %Genxx%Gen, (for the default copy of the file)

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

Regeneration After Testing

To make changes to the application, you must change the CA Gen model, not the generated code, to generate the updated application. After changing the model, regenerate that portion of the application on the development platform.

When the change does not update the load module definition, only select the necessary load module components for regeneration instead of the complete model. Move the regenerated components to the target system, overlaying the old versions of the components, and rebuild the load module. This saves time and resources required to split a remote file whose definition has not changed.

Note: Verify the file names, especially suffixes, before overlaying files. If file names differ, change the name of the new file to match the old one so the MAKE command procedure used during the rebuild searches for the correct file.

If a load module definition changes or most of its components change, generate a new remote file, move it to the target system, and rebuild the application.

After testing an application, you can regenerate it without the trace code and install it for production use.

Setting Environment Variables

Before running C block mode applications and C Servers, set the following environment variables:

Environment Variable	Description
AEPATH*	Defines a set of directories containing transaction routing tables, AEENV files, application.ini files, and load modules. Load modules are stored in the models' source directory or \inqload subdirectory. These directories specify the search order for routing tables and load modules. As a requirement, AEPATH must contain the CA Gen Runtime path referenced by %GEN*xx*%Gen*yy* to locate the p3270key file. **Note:** *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*. **Note:** *yy* refers to the compiler version folder which will be used when executing the Implementation Toolset. For example, VS100\amd64. AEPATH supports test and production directories. Example: set AEPATH <model directory>;%GEN85%Gen\VS100\amd64 For more information about using the application.ini file, see the appendix Using the Application.ini File.
IEFGXTP*	Defines the directory that contains the codepage translation files. Example: set IEFGXTP %GENxx%\Gen\Translat\ **Note:** *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.
IEFH*	Defines the home directory for the Implementation Toolset that contains the CA Gen libraries and binaries, and identifies the location of the p3270key used by the AEFN and the AEFAD. **Note:** Set IEFH when installing the Implementation Toolset or the GUI Runtime. Example: set IEFH=%GENxx%\Gen **Note:** *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.
PATH	Defines a search path that enables locating the DBMS and other executables without specifying the full directory path. The variable entry must include the CA Gen directory.

Note: * Not used for C GUI applications.

GENENV.BAT

The batch file Genenv.bat helps you set the PATH environment variable based on which Microsoft Visual Studio compiler you have built your application against and will run with.

To execute the Genenv.bat file, run the following command:

```
%GENxx%Gen\genenv.bat [bits]
```

[bits]

This value can be 32 or 64. If [bits] is not provided, the default is 32.

The execution of Genenv.bat prepends a set of Microsoft Visual Studio paths and either %GENxx%Gen\VSabc or %GENxx%Gen\VSabc\amd64 to the PATH environment variable.

Note: VSabc refers to the supported version of Visual Studio. Replace VSabc with VS100 for Visual Studio 2010 and VS110 for Visual Studio 2012. xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Note: If the application execution fails, see the Troubleshooting chapter (see page 79).

Application Execution Facility

The Application Execution Facility (AEFN) is a screen-based utility used as an interface between the operating system and generated CA Gen applications. The following diagram illustrates the AEFN screen:

```
-------- ------------------------------------------------------------

trancode clear-screen-options

                                CA Gen



               CA Gen Application Execution Facility

                  Transaction Processing Mode

pf3 = Terminate AEF
```

Note: To execute applications through the AEFN, build them with the profile token OPT.CMD_FORM set to NO. For more information, see the *Build Tool User Guide*.

Invoke the Application Execution Facility

Invoke the AEFN from a command window, using the following command syntax:

```
aefn [options]
```

The following table defines the AEFN command options:

Option	Description
-e [key]	Optional application end key. Default is PF3.
-m [char]	Application mode; either *t* (transactional) or *i* (interactive). Default is *t*.
-l	Display login screen to collect userid & password.
-w [password]	Password for login. Default is PASSWORD. Use with the –l option.
-t [number]	Value between 0 and 31 that activates an internal AEFN program trace. 15 is most useful level. The option creates a file in the %USERPROFILE%\AppData\Local\CA\Gen xx\logs\server directory containing the trace data. The filename is lg-<procname>-<procid>.log, where <procname> is AEFN or the name of the executing load module, and <procid> is the AEFN or executing load module's process number id.
-o [filename]	Name of file to contain dumped screen images.
-s [filename]	Name of the script file to record the current input activity. Use the –p option at a later time to replay the script file.
-p [filename]	Name of a script file containing previously recorded input activity. Use this option to activate the playback feature.
-d [number]	The number of seconds to delay input messages when replaying the script file. You must include this parameter to activate the replay feature. For example, -d 0 activates replay with 0 seconds delay between message inputs. Use with the –p option.
-b	Specifies to display a blank screen during playback. Use with the –p option.
-help	Displays usage information

Note: Applications built as transactional are invoked through the AEFN. Client, server, or command line applications are not invoked through the AEFN.

Function key assignments for the AEFN may differ for each terminal type. The AEFN uses a keyboard mapping file, p3270key, to assign values for each function key and special purpose key available for generated applications on the target system.

The AEFN screen accepts trancodes and clear screen input. The aeenv file in each application's directory defines the mapping of trancodes to load modules. The environment variable AEPATH lists each application's directory for the AEFN to locate the aeenv files. The AEFN reads the trancode entered, and invokes the associated load module. When the load module terminates, control returns to the AEFN.

Key Mapping

When the application starts through the AEFN, keyboard mapping translates terminal keystrokes to the function key structure the application uses. Appropriate key mapping should be established before executing applications.

Application Execution Facility Record and Playback Feature

The AEFN can record and playback a user's interaction with the application. The recording, or script, can be played back later if the database is unchanged.

Capabilities

The AEFN record and playback features include the ability to:

- Record scripts and play them back for applications that flow between multiple load modules
- Specify playback speed
- Create platform-independent scripts
- Read and edit scripts with any text editor

During recording, as the user interacts with a CA Gen application, the keystrokes are stored in the script file and control sequences are mapped to AEFN p3270 emulator commands. The emulator commands are stored in the script file in a readable form. Each command begins on a new line and is enclosed in pound signs (#).

The following is an example of a script:

```
#PF2#
#tab#
40#eraseof#
#newIn#
Bonny
Smith#newIn
#123 Home
st.#newIn#
Richara#left#
dson#newIn#
tx78341#pf2#
#PF12#
#clear#
```

Use the AEFN p3270 emulator help screen (Esc h) to determine valid AEFN p3270 emulator command names. This is an example of the help screen:

```
------------------------------------------------------------------------

home    -> Home         | PF6   -> F6          | bspace  -> Backspace |

Enter   -> Numpad Enter | PF7   -> F7          | refresh -> Ctrl W    |

Clear   -> Ctrl Home    | PF8   -> F8          |

PA1     -> Alt F4       | PF9   -> F9          |

PA2     -> Alt F5       | PF10  -> F10         |

PA3     -> Alt F6       | PF11  -> F11         |

Help    -> Esc h        | PF12  -> F12         |

Playback-> Esc p        | PF13  -> shift F1    |

Togl stl-> Esc l        | PF14  -> Shift F2    |

Hidden  -> Esc i        | PF15  -> Shift F3    |

Tab     -> Tab          | PF16  -> Shift F4    |

Bktab   -> shift Tab    | PF17  -> Shift F5    |

Insert  -> Insert       | PF18  -> Shift F6    |

Delete  -> Delete       | PF19  -> Shift F7    |

Reset   -> Esc Esc      | PF20  -> Shift F8    |

Eraseof -> Ctrl Delete  | PF21  -> Shift F9    |

Chg atr -> Esc d        | PF22  -> Shift F10   |

override-> Alt z        | PF23  -> Shift F11   |

show cod-> Esc t        | PF24  -> Shift F12   |

PF1     -> F1           | left  -> left arrow  |

PF2     -> F2           | right -> right arrow |

PF3     -> F3           | up    -> up arrow    |

PF4     -> F4           | down  -> down arrow  |

PF5     -> F5           | newln -> Enter       |

------------------------------------------------------------------------
```

Features

Use the AEFN record and playback feature to:

- Regression test CA Gen applications or AEFN after making software changes.
- Automatically initialize a user's dialog to a screen or menu. A user can customize the session using the AEFN p3270 emulator Playback and Hidden toggle commands from the keyboard or playback file. The Hidden command from the playback file prevents screen display when the initial AEFN program startup includes -b. The Playback command from the playback file ends the replay and begins AEFN p3270 emulator operation.
- Automatically terminate a user's dialog. The keyboard Playback command (Esc p) ends AEFN p3270 emulator operation and resumes replay from the playback file. The Hidden command from the keyboard (Esc i) toggles the screen display.

Limitations

The AEFN record and playback feature includes the following limitations:

- Always delete the recording file to start fresh because AEFN scripts always append to the specified file.
- The database must remain unchanged between the time the script begins and the time it is played back.

Invoke the AEFN for Record/Playback

To start recording, enter the following command:

```
aefn -s [filename]
```

To initiate script playback, enter the following command:

```
aefn -d 0 -p [filename]
```

Application Testing

The following sections describe application testing philosophy and the requirements to use the Diagram Trace Utility.

Diagram Trace Utility

Use the Diagram Trace Utility to test a generated application before moving it to a production environment. It steps through the application as it executes, allowing you to view the CA Gen model elements used to build the application, such as action diagram statements, as the generated program executes. To view the model elements, you must make certain selections that generate additional code when generating remote files.

When testing an application, the procedure steps and action blocks generated with trace communicate with the Diagram Trace Utility. The application screens update after control returns from the Diagram Trace Utility.

Before testing an application, you must build certain application components with the Build Tool:

- The application database
- RI trigger logic
- Operations libraries
- All load modules

Enable the Diagram Trace Utility

Follow these steps:

1. Build the test application generated with trace.
2. Invoke the Diagram Trace Utility on any Windows system, including the same system:
   - Launch the Diagram Trace Utility from 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 starts listening on port 4567. Change the default port in the Diagram Trace Utility.
3. Set the trace environment variables in the application.ini file.
4. Set the AEPATH environment variable when invoking the application through the AEFN.

More information:

Using the Application.ini File (see page 89)

Access and Use the Diagram Trace Utility

You can use the Diagram Trace Utility to test all generated C applications on a Windows system. Each application requires transaction codes to map to the correct procedure steps and action blocks.

The following table lists the C applications available, and how to invoke them:

C Application	Invocation Method	How Trancode is Obtained
GUI Windowed	Command Line or Windows Explorer	As a parameter when invoked from the command line. When invoked from the Windows Explorer, the first trancode stored within the executable is used.
Distributed Processing Client (GUI Client)	Command Line or Windows Explorer	As a parameter when invoked from the command line. When invoked from the Windows Explorer, the first trancode stored within the executable is used.
Distributed Processing Server	AEFAD or WebSphere MQ	From the AEENV file.
Block Mode Command Line	Command Line	As a parameter when invoked, then from the AEENV file during execution.
Block Mode Interactive	AEFN	From the AEENV file.

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

Multi-User Diagram Trace Utility Support

When a remote server application starts, one copy of the application.ini file is available for that application. Setting the trace environment variables in the application.ini file only allows one Diagram Trace Utility the ability to trace the servers in the server application.

In a test environment with multiple testers working with the same server application, testers need to debug from multiple client workstations. To do so, the environment needs multiple Diagram Trace Utilities.

Overriding the default trace environment variables

A client transaction can transfer the host and port of the client executing the transaction, enabling the server to establish communication with the Diagram Trace Utility running on the client workstation.

Follow these steps:

1. Build the server application that was generated with trace.
2. Leave the trace environment variables in the application.ini file on the server application's model directory commented out.
3. Invoke the TP monitor that will manage the server application, TE, Tuxedo, or MQ Series.
4. Invoke the Diagram Trace Utility on the client workstation.
5. Edit the application.ini file in the client application's executables directory.
6. Uncomment and set the trace environment variables. Use 'localhost' for the TRACE_HOST variable.
7. Execute your client application.

   The Diagram Trace Utility on the client workstation traces the server for each transaction initiated from that client.

Regenerating Remote Files After Testing

The code generated on the CA Gen workstation for testing with trace includes features, such as trace calls, that are only needed during testing. These features increase the size of the load module significantly, and impact the application's performance. Even if no changes are required as a result of the tests performed, you must regenerate the load module without trace before it is placed into production.

Application Production

When system maintenance is required, make CA Gen model changes on the workstation or mainframe, regenerate the changed elements into remote files, and move the files to the target system. This allows enhancements and modifications to implemented systems without changing the generated code.

After an application is tested and ready for production, move it to the proper environment and introduce it into production. Each organization using the Implementation Toolset may perform these tasks differently. Some elements are common to all installations.

To run a generated application in production requires the application, the DBMS the application uses, and the AEFN. The AEFN serves as an operating environment to handle communication between the operating system and the generated application.

Many CA Gen operating environment elements can be customized for the target system. The CA Gen runtime user exits allow access to certain system functions, such as retrieving a user ID or handling errors that may be changed to fit a specific target system configuration. The user exits are source language routines in which to add customized code. When compiled and placed in the proper library, the load module's dialog manager accesses them as the generated application executes.

Test Changes to a Production Application

To test changes to a production application, isolate the test environment from the production environment by installing the remote files in a different location and generating a test database using a different name than the production database name. This ensures that the remote files generated for test do not overwrite other remote file components or access the production database.

Appendix A: User Exits

Note: The Implementation Toolset (IT) must be installed on your target system before modifying runtime user exits. The installation process copies the runtime user exits to your target system and stores them in the %GENxx%Gen directory.

CA Gen supports certain system functions, such as retrieving a user ID, error handling, and site-specific security so users do not have to consider them when programming their application. These functions are user exits because users can modify the source. User exits are delivered in source and object form.

This section contains the following topics:

Runtime User Exits

Runtime user exits are standard routines that reside on the target system and allow applications generated by CA Gen to access the system features. All generated applications can access the routines without including the routines as part of the application. These user exits can supply basic functionality, or you can customize them for the target system.

All runtime user exits are provided in source and object format and loaded onto the target system during IT installation. You can use the object format runtime user exits without change. They are ready to link into applications as you compile the application. The source code is provided in all supported languages, so you can modify runtime user exits to meet your site requirements. After completing the modifications, compile the modified user exit, and overwrite the existing version with your new one using the following scripts:

- For BlockMode and Server User Exits for Visual Studio 32-bit use %GENxx%Gen\VSabc\mkexits.bat
- For BlockMode and Server User Exits for Visual Studio 64-bit use %GENxx%Gen\VSabc\amd64\mkexits.bat
- For GUI User Exits for Visual Studio 32-bit use %GENxx%Gen\VSabc\mkexitsn.bat

Note: VSabc refers to the supported version of Visual Studio. Replace VSabc with VS100 for Visual Studio 2010 and VS110 for Visual Studio 2012. xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

You do not need to relink the applications that use the recompiled user exit for the change to be effective.

When migrating to the latest CA Gen environment, it may be necessary to recompile your user exits. This may be due to a number of reasons, including changes to the calling interface, compiler upgrades, and third party product upgrades.

Note: For release specific instructions, see *Release Notes*.

There is a set of user exits for each CA Gen Runtime supported on Windows, those for GUI applications and those for block mode applications. Batch files are provided to assist in building the runtime user exits for each runtime environment listed in the following sections.

The following table lists the set of runtime user exits for the CA Gen Runtimes.

Application Type	Provided as
GUI applications	Wrexitn.c
Block mode Applications	tir*.c files

User Exits Provided by CA Gen

There are also user exits for z/OS, UNIX, and Linux as part of Host Construction and on CA Gen Target systems. Customized user exits belong to a specific platform. Do not upload or move them to other target systems. Target operating system constraints, the compiler configuration or the site procedures might limit specific processing.

GUI Runtime User Exits

The following table summarizes the functions available through the GUI runtime user exits:

Name	User Exit Description
WRDRTL	Default Retry Limit Exit
WRURTL	Ultimate Retry Limit Exit
WRGLB	Globalization Exit
WRSYSID	System ID Exit
WRUSRID	User ID Exit
WRTERMID	Terminal ID Exit
WRUPPR	Uppercase Translation Exit
WRSECTOKEN	Client Security Token User Exit
WRSECENCRYPT	Client/Server Encryption Exit
WRSECDECRYPT	Client/Server Decryption Exit
WRSRVERROR	Client/Server Flow Failure Exit
WRDEFAULTYEAR	Century Default Exit
WRASYNCSRVRERROR	Client/Server Asynchronous Flow Server Failure Exit

Note: For more information about GUI runtime user exits, see the *User Exit Reference Guide*.

Block Mode Runtime User Exits

The following table summarizes the functions available through the user exits for generated block mode applications:

Name	Description
TIRDCRYP	Decrypt User Exit
TIRNCRYP	Encrypt User Exit
TIRDLCT	User Dialect User Exit
TIRDRTL	Default Retry Limit User Exit
TIRURTL	Ultimate Retry Limit User Exit
DBCONNCT	Database connection User Exit. There is one user exit routine for each supported database: ODBC, Oracle, and DB2.
DBCOMMIT	Database commit User Exit. There is one user exit routine for each supported database: ODBC, Oracle, and DB2.
DBDISCNT	Database disconnect User Exit. There is one user exit routine for each supported database: ODBC, Oracle, and DB2.
TIRELOG	Server Error Logging User Exit
TIRHELP	Help Interface User Exit
TIRUPDB	MBCS Uppercase Translation User Exit
TIRUPPR	Uppercase Translation User Exit
SRVRERROR	Server to Server Error User Exit
TIRMTQB	Message Table User Exit
TIRSECR	Security Interface User Exit
TIRSECV	Server Security Validation User Exit
TIRSYSID	System ID User Exit
TIRUSRID	User ID User Exit
TIRTERMA	User Termination User Exit
TIRXLAT	National Language Translation User Exit
TIRXINFO	Locale Information User Exit
TIRYYX	Date User Exit

Block mode runtime user exits are rebuilt into the DLL AEUEXITxxN.DLL using the command procedure %GENxx%Gen\VSabc\mkexits.bat for Visual Studio 32-bit, or %GENxx%Gen\VSabc\amd64\mkexits.bat for Visual Studio 64-bit.

Note: VSabc refers to the supported version of Visual Studio. Replace VSabc with VS100 for Visual Studio 2010 and VS110 for Visual Studio 2012. xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Note: For more information about block mode runtime user exits, see the *User Exit Reference Guide*.

Appendix B: Rebuilding DBMS DLLs and Executables

CA Gen supplies a set of DBMS DLLs to use with block mode applications and servers and executable stubs to use with GUI applications. These DLLs and executables provide default connect, disconnect, commit, rollback, and error behavior for the supported databases.

Note: For the list of supported data base versions, see the *CA Gen Technical Requirements*.

Customers with requirements that differ from those in the *CA Gen Technical Requirements* for DBMS versions can rebuild the appropriate DLLs or executable stubs using supplied .bat and .mak files. The following sections describe rebuilding the DLLs and executable stubs.

This section contains the following topics:

DBMS DLLs

DBMS DLLs support block mode applications and servers, and are rebuilt using the batch procedure %GENxx%Gen\VSabc\mkdbs.bat for Visual Studio 32-bit or %GENxx%Gen\VSabc\amd64\mkdbs.bat for Visual Studio 64-bit.

Note: VSabc refers to the supported version of Visual Studio. Replace VSabc with VS100 for Visual Studio 2010 and VS110 for Visual Studio 2012. xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

The following table lists the set of DLLs that can be rebuilt:

File Name	Supports
aecdb2.dll	DB2
aecodb.dll	ODBC
aecora.dll	Oracle

DBMS DLL Rebuild

The file mkdbs.bat is used to rebuild the appropriate DLL when the DBMS version is different than what is specified in the *CA Gen Technical Requirements*.

Follow these steps:

1. Launch an MS-DOS Command window.
2. Change the current directory to the directory that contains the appropriate batch file MKDBS.BAT, %GENxx%Gen\VSabc for Visual Studio 32-bit or %GENxx%Gen\VSabc\amd64 for Visual Studio 64-bit.

   Note: VSabc refers to the supported version of Visual Studio. Replace VSabc with VS100 for Visual Studio 2010 and VS110 for Visual Studio 2012. xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.
3. Run the command:

   `mkdbs.bat <dbms>`

   <dbms> is *one* of:

   - ORACLE—Oracle 11g is the default
   - ORACLE10—Oracle 10g
   - DB2
   - ODBC

Note: The PATH, LIB, and INCLUDE environment variables need to contain proper entries for the DBMS chosen.

Before executing mkdbs.bat, you must set the appropriate environment variables for the specific DBMS you are working with. The following table lists the set of environment variables required for each DBMS:

DBMS	Environment Variables	Description
DB2	DB2PATH	Base Directory for DB2
	AEDB	Database to Connect to
	AEUSER (optional)	Userid of Database to Connect to
	AEPASSWORD (optional)	Password of Database to Connect to
ODBC	N/A	
Oracle	ORACLE_HOME	Base Directory for Oracle

Note: If you choose to use an alternate version of Oracle, there is built-in logic within the mkdbs.bat file to handle Oracle 11g, the default version, and Oracle 10g. When working with Oracle 10g, you must pass the parameter ORACLE10 when executing mkdbs.bat.

Example:

```
%GENxx%Gen\mkdbs ORACLE10
```

Altering the DBMS Version

To alter the DBMS version they will use with the block mode and server runtimes, you need to make the following hand-edits in the Windows Runtime environment:

Follow these steps:

1. Update the appropriate make procedure mkdbs.bat:

   The mkdbs.bat file contains a list of DBMS-specific libraries linked into the appropriate DBMS library supplied with CA Gen. Update this set of libraries for the correct DBMS version.

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

2. Change the current directory to the directory that contains the appropriate batch file MKDBS.BAT, %GENxx%Gen\VSabc for Visual Studio 32-bit or %GENxx%Gen\VSabc\amd64 for Visual Studio 64-bit.

   Note: VSabc refers to the supported version of Visual Studio. Replace VSabc with VS100 for Visual Studio 2010 and VS110 for Visual Studio 2012. xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

3. Run mkdbs.bat:

   The mkdbs.bat procedure updates the database library and the DDL loader for the DBMS.

4. Update the Build Tool Profile:

   After your Windows Runtime environment is adjusted for an alterative DBMS version, you need to reflect these changes in your Windows Build Tool profile before building any applications.

The following tokens must be modified appropriately:

Token Key	Value
LOC.DBLIB	DBMS library path
LOC.DBPATH	DBMS binary path

Token Key	Value
LOC.DBINCLUDE	DBMS include path
OPT.DBLIB	DBMS library filename
OPT.DBSQLLIB	DBMS SQL library filename
OPT.DBPCC	DBMS precompiler filename
OPT.DBPCCFLAGS	DBMS precompiler flags

Note: For more information, see the *Build Tool User Guide*.

Database Loader Rebuild

When mkdbs.bat executes, it executes the appropriate makeddl.bat in %GENxx%Gen\VSabc\DDL (for Visual Studio 32-bit) or %GENxx%Gen\VSabc\amd64\DDL (for Visual Studio 64-bit).

Note: VSabc refers to the supported version of Visual Studio. Replace VSabc with VS100 for Visual Studio 2010 and VS110 for Visual Studio 2012. xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

This procedure rebuilds the database loader for the selected DBMS. The Build Tool uses the database loader, identified as TI<dbms>DDL.EXE, when performing a build step on the database ICM module. The rebuild of the selected DBMS database loader ensures that all DBMS-specific DLLs and EXEs are built with the same DBMS version.

The following table lists the set of executables that can be rebuilt:

File Name	Loads Database Tables For
tidb2ddl.exe	DB2
tiodbddl.exe	ODBC
tioraddl.exe	Oracle

DBMS Stub EXEs

DBMS executable stubs support GUI applications and are rebuilt using a set of make procedures. The following table lists the set of executable stubs that can be rebuilt:

File Name	Supports	Make Procedure
Stubdb2n.exe	DB2	Stubdb2n.mak

File Name	Supports	Make Procedure
Stubn.exe	No DBMS (default)	Stubn.mak
Stubodbn.exe	ODBC	Stubodbn.mak
Stuboran.exe	Oracle	Stuboran.mak

DBMS Stub EXE Rebuilding

Each of the stub<dbms>n.mak files are located in the %GENxx%Gen\VSabc directory (for Visual Studio). Some DBMS choices, require that the particular stub<dbms>n.mak file execute before building the application with the Build Tool.

Note: VSabc refers to the supported version of Visual Studio. Replace VSabc with VS100 for Visual Studio 2010 and VS110 for Visual Studio 2012. xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Follow these steps:

1. Launch an MS-DOS Command window.
2. Change the current directory to the directory that contains the appropriate makefile STUB<dbms>N.MAK, %GENxx%Gen\VSabc for Visual Studio.

   Note: VSabc refers to the supported version of Visual Studio. Replace VSabc with VS100 for Visual Studio 2010 and VS110 for Visual Studio 2012. xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.
3. Run the command:

   ```
   nmake –nologo –s -f stub<dbms>n.mak all
   ```

Note: The PATH, LIB, and INCLUDE environment variables must contain entries for the appropriate DBMS chosen.

Before executing any of the stub<dbms>n.mak files, you must set the appropriate environment variables for the specific DBMS that you are working with, similar to mkdbs.bat. The following table lists the set of environment variables required for each DBMS choice:

DBMS	Environment Variables	Description
DB2	AEDB	Database to Connect to
	AEUSER (optional)	Userid of Database to Connect to
	AEPASSWORD (optional)	Password of Database to Connect to
ODBC	N/A	
Oracle	ORACLE_HOME	Base Directory for Oracle

Altering the DBMS Version

To alter the DBMS version that they will use with the GUI Runtimes, the following hand-edits must take place in the Windows Runtime environment:

Follow these steps:

1. Update the appropriate make procedure stub<dbms>n.mak:

   The stub<dbms>n.mak file contains a set of DBMS-specific libraries linked to the appropriate DBMS stub executable supplied with CA Gen. This set of libraries must be updated based on the DBMS version.

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

2. Change the current directory to the directory that contains the appropriate makefile stub<dbms>n.mak, %GENxx%Gen\VSabc for Visual Studio.

   Note: VSabc refers to the supported version of Visual Studio. Replace VSabc with VS100 for Visual Studio 2010 and VS110 for Visual Studio 2012. xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

3. Run stub<dbms>n.mak

   The stub<dbms>n.mak procedure updates the database stub executable and rebuilds the DDL loader for the DBMS.

4. Update the Build Tool Profile:

   After adjusting the GUI Runtime environment for an alternate DBMS version, reflect the changes in the Windows Build Tool profile before building applications. Modify the following tokens appropriately:

Token Key	Value
LOC.DBLIB	DBMS library path
LOC.DBPATH	DBMS binary path
LOC.DBINCLUDE	DBMS include path
OPT.DBLIB	DBMS library filename
OPT.DBSQLLIB	DBMS SQL library filename
OPT.DBPCC	DBMS precompiler filename
OPT.DBPCCFLAGS	DBMS precompiler flags

Note: For more information, see the *Build Tool User Guide*.

Database Loader Rebuild

When stub<dbms>n.mak executes, it executes the appropriate makeddl.bat in %GENxx%Gen\VSabc\DDL (for Visual Studio).

Note: VSabc refers to the supported version of Visual Studio. Replace VSabc with VS100 for Visual Studio 2010 and VS110 for Visual Studio 2012. xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

This procedure rebuilds the database loader for the selected DBMS. The Build Tool uses the database loader, identified as TI<dbms>DDL.EXE, when performing a build step on the database ICM module. The rebuild of the selected DBMS database loader ensures that all DBMS-specific .DLLs and .EXEs are built with the same DBMS version.

The following table lists the set of executables that can be rebuilt:

File Name	Loads Database Tables For
tidb2ddl.exe	DB2
tiodbddl.exe	ODBC
tioraddl.exe	Oracle

Special Considerations

The MS/SQL database loader, TIS95DDL.EXE, is no longer associated to a matching DBMS DLL or DBMS Stub EXE. Because Microsoft no longer supports Embedded SQL; generated applications using MS/SQL as the Technical Design can only use an ODBC access method. The database tables are still in an MS/SQL database that TIS95DDL.EXE loads.

Follow these steps:

1. Change the current directory to the directory that contains the appropriate bat file makeddl.bat, %GENxx%Gen\VSabc for Visual Studio 32-bit or %GENxx%Gen\VSabc\amd64 for Visual Studio 64-bit.

   Note: VSabc refers to the supported version of Visual Studio. Replace VSabc with VS100 for Visual Studio 2010 and VS110 for Visual Studio 2012. xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

2. Run the following:

   ```
   makeddl.bat MSSQL
   ```

Note: You must change the current directory to the directory that contains the batch file makeddl.bat, by default, ddl subdirectory in the CA Gen installation directory.

You can only rebuild this executable with the currently supported version of MS/SQL or any future releases that support SQL-DMO.

Appendix C: Troubleshooting

This appendix lists common problems that occur when using the Implementation Toolset (IT). It is divided into these two sections:

- Troubleshooting tables for AEFN, DBMS, and application execution problems
- General technical considerations

The following sections list troubleshooting information for target system implementation processes.

This section contains the following topics:

AEFN Troubleshooting

Problem	User Action
Nothing happens after entering a transaction code	■ Check %AEPATH% ■ Check for aeenv file with trancode in %AEPATH%. ■ Ensure you have execute privileges for the load module in %AEPATH%\inqload.
`OOPS OOPS OOPS ...` message when the AEFN starts.	■ Ensure %IEFH%\p3270key properly defines the TERM environment variable.
`Term setup file missing/wrong.` message when the AEFN starts.	■ Ensure the IEFH environment variable is correctly set. ■ Ensure %IEFH%\p3270key properly defines the TERM environment variable. ■ Ensure you have read privileges for the p3270key file.
`Command not found.` Message when the AEFN starts	■ Ensure PATH includes %IEFH%. ■ Ensure you have execute privileges for %IEFH%\aefn.exe.

DBMS Troubleshooting

Problem	User Action
Unable to connect to Oracle.	■ Ensure that Oracle is running. ■ Ensure your ORACLE_SID environment variable is correctly set.
A DBMS upgrade has just been performed.	■ Ensure the IT scripts have the correct DBMS link libraries after upgrade. ■ Ensure that DBMS directory references in IT scripts are correct after upgrade. ■ Ensure that pre-compilers have not changed. ■ Ensure mkdbs.bat was executed to update the DBMS-specific link library CA Gen supplies. ■ Ensure your application restarted.
The following Oracle DBMS error was received: Invalid id/password	■ Ensure the DBNAME in the target definition has connect and resource capabilities to Oracle.

Application Execution Troubleshooting without Using the Application Execution Facility

Problem	User Action
Nothing happens after entering a transaction code and .	■ Check %AEPATH%. ■ Check for aeenv file with trancode in %AEPATH%.
When executing 32-bit Application built with Visual Studio in command window, user receives error window stating "<executable> has encountered a problem and needs to close."	Prepend %GENxx%Gen\VSabc to PATH. By default PATH includes %GENxx%Gen, but not the \VSabc folder.
64-bit application servers built with MQseries fail to start with error "Cannot find MQICF80N.DLL".	Ensure MQSeries' \bin64 directory is in PATH, and MQSeries' \lib64 directory is in LIB.

Problem	User Action
Receive "733 - Allocation Abend" when starting 64-bit application server using the TE.	Prepend %GENxx%Gen\VSabc\amd64 to PATH. By default PATH includes %GENxx%Gen, but not the \VSabc\amd64 folder used by 64-bit applications.
When executing 64-bit server application in command window, user receives error window stating "<executable> - Application Error. The application was unable to start correctly (0xc000007b)."	Prepend %GENxx%Gen\VSabc\amd64 to PATH. By default PATH includes %GENxx%Gen, but not the \VSabc\amd64 folder used by 64-bit applications.

Note: VSabc refers to the supported version of Visual Studio. Replace VSabc with VS100 for Visual Studio 2010 and VS110 for Visual Studio 2012. xx refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Diagram Trace Utility Troubleshooting

The following table provides troubleshooting information regarding use of the Diagram Trace Utility with applications generated with trace enabled.

Problem	User Action
Application runs without communicating with the Diagram Trace Utility	■ Ensure the following environment variables are uncommented and set in the application.ini file: ■ TRACE_ENABLE ■ TRACE_HOST ■ TRACE_PORT ■ Ensure the environment variable AEPATH is set correctly when running your application through the AEFN. ■ Make sure the Diagram Trace Utility started. ■ Make sure the Diagram Trace Utility has at least one breakpoint set or has the preference *Suspend on initial* entry selected. When neither are set, it appears as if the application is not communicating with the Diagram Trace Utility.

General Technical Tips

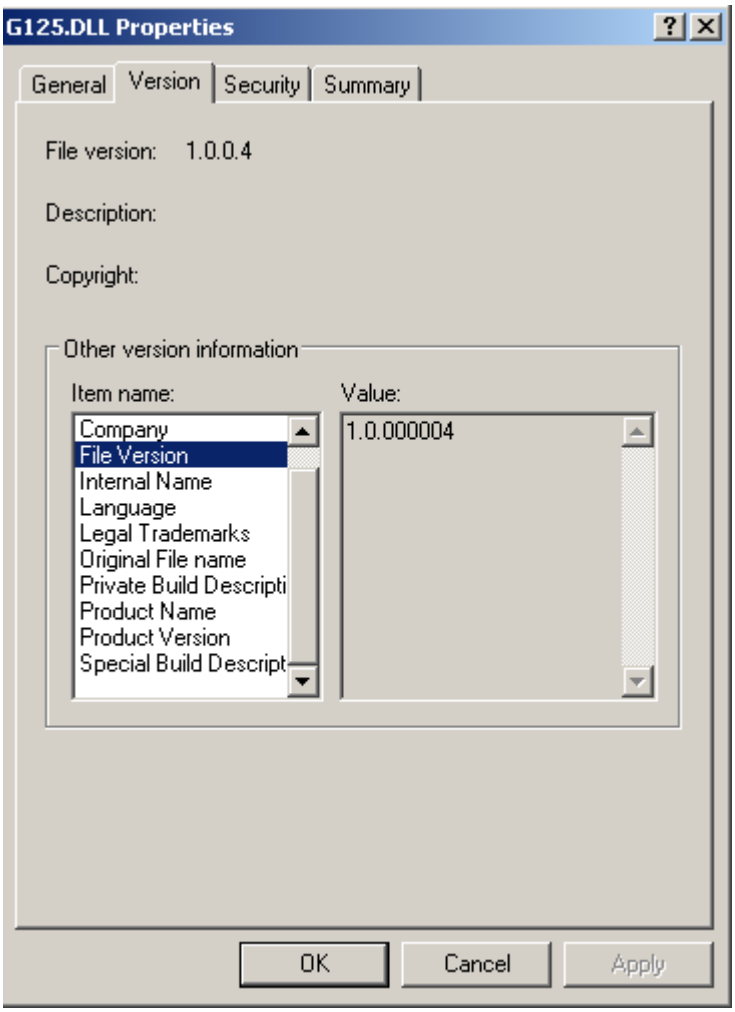
These general tips help reduce problems:

- It is important to closely adhere to guidelines for software version requirements, for example, DBMS and C versions.
- Separate AEPATH paths and directories with a semicolon (;), not a colon (:).
- When generating a cooperative application, generate the client and server separately even if the client and server are targeting the same machine.

Appendix D: Application Versioning

Applications that are built on the Windows platform contain a set of version properties that can be set when the application is linked.The Implementation Toolset includes a set of files to link this version information into the applications generated with CA Gen, particularly GUI DLLs, Block-mode and Sever EXEs, Cascade DLLs, and 0perations library DLLs.

This version information is visible in the CA Gen generated DLLs or EXE's Properties window in the Version tab as illustrated below:



In order to modify the version information displayed for CA Gen generated applications, the following files are delivered:

- **generic.rc**—This is a Windows Resource file, containing several resource definitions that are utilized by the Resource Compiler. When linked into the CA Gen generated application, the fields available within the Version tab of the Properties Dialog are populated.
- **application.h**—This header file provides a set of #define macros that are then used by the resource definitions provided in the generic.rc file. This file can be customized.
- **applno.bat**—This Windows batch procedure is executed during the building of a CA Gen generated application, and is responsible for incrementing an application build count and refreshing the application's build timestamp. The results of the build count and timestamp are stored in an application specific header file that is also made available to the generic.rc file.

This section contains the following topics:

How Application Versioning Works (see page 84)

How Application Versioning Works

When an application is built for the first time, 2 files are copied and/or created as part of the application build. These files are then used repeatedly when rebuilds of the same application occur. Each of these files are explained below:

application.h

Application.h contains a number of #defines that refer to fields displayed through the Version tab of the Properties window. The customer can edit the #defines in this file to customize their application's version information.

The first time an application is built, the master version of application.h is copied from the CA Gen installation directory into the model directory. Once this file is copied, it can be modified in order to customize particular version information for all future builds.

***_buildver.h**

The first time an application is built, a new header file is generated to contain a build count and the current timestamp. These values are used to refer to fields also being displayed through the Version tab of the Properties window. Each time the application is rebuilt, the build count is incremented, and the timestamp is updated. This file should only be modified to set the build count to a specific numeric value.

There is one copy of this file for each application being built.

When an application build occurs, applno.bat will read the existing *_buildver.h file, increment the build count, and then rewrite this file with the updated build count and timestamp. The resource file generic.rc, which contains #includes statements for application.h and *_buildver.h, is then compiled and linked into the application. The version information that is contained in these header files are now loaded as resources in the application.

Once an application has been built, you can observer the version information in the Version tab from the Properties dialog.

The following table lists the fields displayed in the Version tab of the Properties window for a customer's application, with a mapping to the #defines and the header file they are in. There are 2 sections in the Version tab.

The top section contains 3 fields:

Displayed Field	Description of Field	Mapped #define	Header stored in
File Version	File version of Product, in the format X.X.X.X, appended by the incremented application build number. The starting value is "1.0.0.1"	PRODUCT_VERSION 2, BUILDVERS2	application.h and *_buildver.h
Description	Description of application	CG_FILEDESCRIPTIO N	application.h
Copyright	Copyright information	LEGAL_COPYRIGHT	application.h

The lower section contains a group box titled "Other Version Information", and contains the following fields:

Field	Description	Mapped # defines	Where found
Comments	Can be used for any reason	COMMENTS	application.h
Company	Company Name	COMPANY_NAME	application.h
File Version	File version of file, in the format X.X.X, appended by the incremented application build number. The starting value is "1.0.000001"	FILE_VERSION, BUILDVERS	application.h and *_buildver.h

Field	Description	Mapped # defines	Where found
Internal Name	Internal Project name	CG_INTERNALNAME	application.h
Language	Language of application	CG_LANGID	application.h
Legal Trademarks	Trademark information	LEGAL_TRADEMARKS	application.h
Original Filename	Name of File	CG_ORIGINALFILENAME	application.h
Private Build Description	Timestamp of latest build of application	BUILDTIME	*_buildver.h
Product Name	Name of Product	PRODUCT_NAME	application.h
Product Version	Product version string, in format "X.X.X". The starting value is "1.0.0"	PRODUCT_VERSION	application.h
Special Build Description	Reserved for PTF numbering	CG_PTFNUMBER	application.h

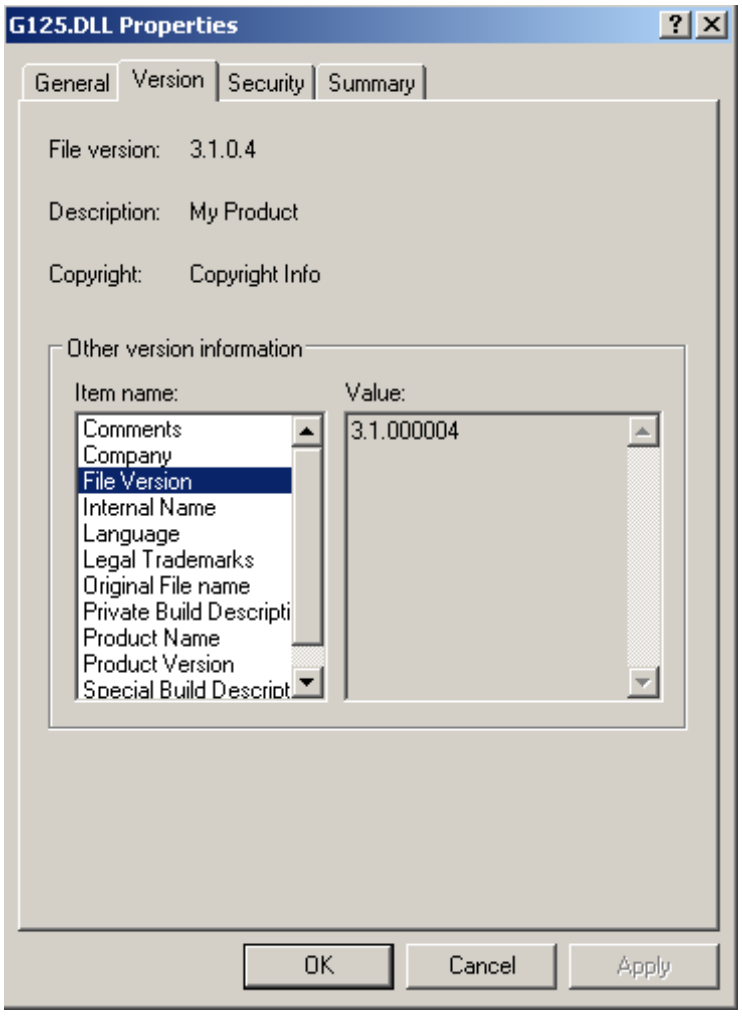
The following example demonstrates the results of using application versioning. The application.h file has been modified as follows:

```
#include "ptfs.h"
#define COMMENTS "My Comments\0"
#define COMPANY_NAME "My Company\0"
#define FILE_VERSION "3.1.0\0"
#define LEGAL_COPYRIGHT "Copyright Info\0"
#define LEGAL_TRADEMARKS "Trademark Info\0"
#define PRODUCT_NAME "My Product\0"
#define CG_FILEDESCRIPTION PRODUCT_NAME
#define PRODUCT_VERSION "2.1.0\0"
#define PRODUCT_VERSION2 2,1,0
#define CG_INTERNALNAME "Alpha\0"
#define CG_ORIGINALFILENAME "File.Ext\0"
#define CG_PTFNUMBER "\0"
```

Also in this example we have built the G125 application 4 times. The G125_buildver.h file will reflect the following values:

```
#define BUILDTIME “Thu 7/29/2010  09:09 AM”
#ifdef BUILDVERS
#undef BUILDVERS
#endif
#ifdef BUILDVERS2
#undev BUILDVERS2
#endif
#define BUILDVERS "00004"
#define BUILDVERS2 00004
```

The following illustration displaying the Version tab of the Properties dialog for the G125.DLL just built:



Appendix E: Using the Application.ini File

This section contains the following topics:

Overview

The application.ini file stores application-specific environment variables in a 'token=value' pairing, similar to the GUIEnvironmentVariables.ini file. CA Gen delivers this initialization file with the C Runtime in the Gen install directory %GENxx%Gen. When a C application is built using the Build Tool, this file is copied, only once, into the model's source directory (referred to by the profile token LOC.CODE_SRC). This copy of the application.ini file is customizable by the user, and will not be overridden.

When a C application executes, including Windows applications, cooperative applications, and block mode applications, the C Runtime library opens and reads the application.ini file in the model's source directory, and sets uncommented environment variables in the application.

Environment Variables in the Application.ini File

The application.ini file contains a set of commented environment variables. To use these environment variables, remove the comment from the variable (";") and modify the value.

The application.ini contains the following environment variables:

Environment Variable	Description	Default
TRACE_ENABLE	Enables communications with the Diagram Trace Utility.	1
TRACE_HOST	Defines the Windows system running the Diagram Trace Utility.	Localhost
TRACE_PORT	Defines the port on which the Diagram Trace Utility listens.	4567

Note: To use the Diagram Trace Utility, generate the C application with 'Trace Enabled'.

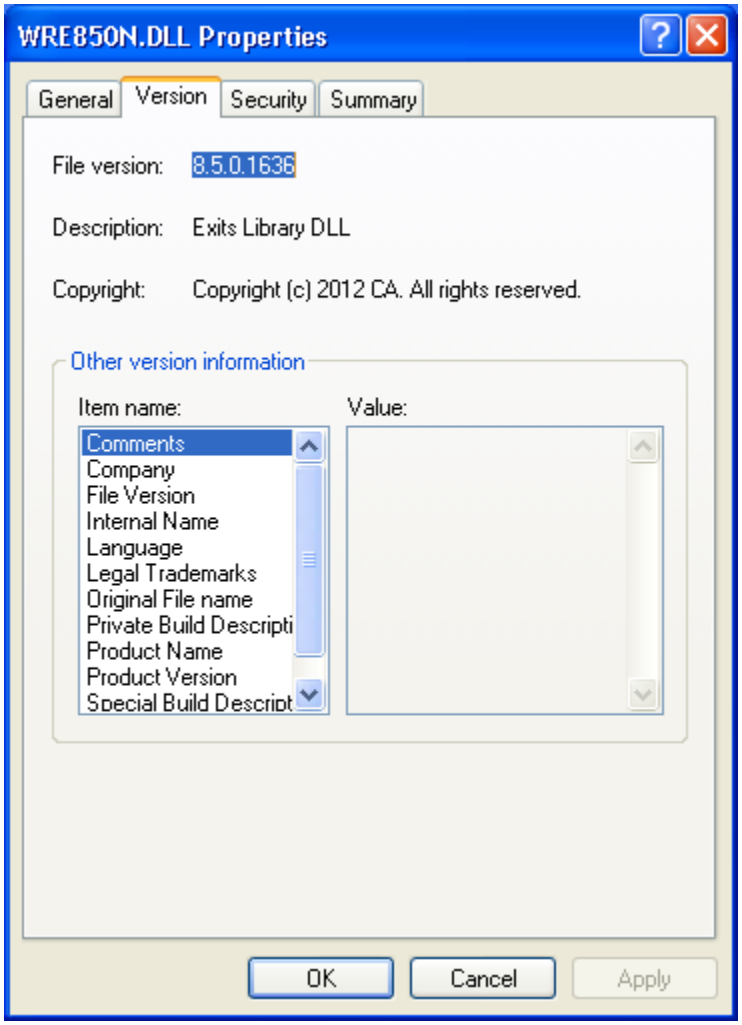
Appendix F: Rebuildable DLL and EXE Build Counts

As part of the Windows IT installation, there are several delivered DLLs and EXEs that are rebuildable by the customer. These DLLs and EXEs include User Exits, DBMS DLLs, DBMS Loader EXEs, and Dialect DLLs. In order to help identify whether or not one of these files have been rebuilt, CA Gen will now increment a build counter that will be visible on the Version tab of the Properties dialog. This feature is similar to that of Application Versioning, but with limited resource property modifications.

When delivered, the DLLs and EXEs that are rebuildable already have a number of fields populated in the Version tab. These fields remain unchanged when rebuilds occur, in order to properly identify the file that was delivered.

There are a few fields that are now modified when a customer rebuilds one of these files, which aid in identifying a rebuild.

The typical Version tab of the Properties dialog from a delivered rebuildable DLL or EXE looks like the following (using the GUI Runtime User Exit DLL wre850n.dll as an example):



As you can see from the illustration, several of the fields are already populated with CA Gen specific information: File Version, Description and Copyright are viewable at the moment (other fields are not currently focused on and therefore not visible).

While keeping most of these fields ‘as is’ when delivered to continue to identify the DLL or EXE, 2 of the fields can be used to identify that the customer has rebuilt a particular DLL or EXE.

Upon the completion of a rebuild of one of these DLLs or EXEs, the Version tab of the Properties Dialog will now have the following 2 fields modified:

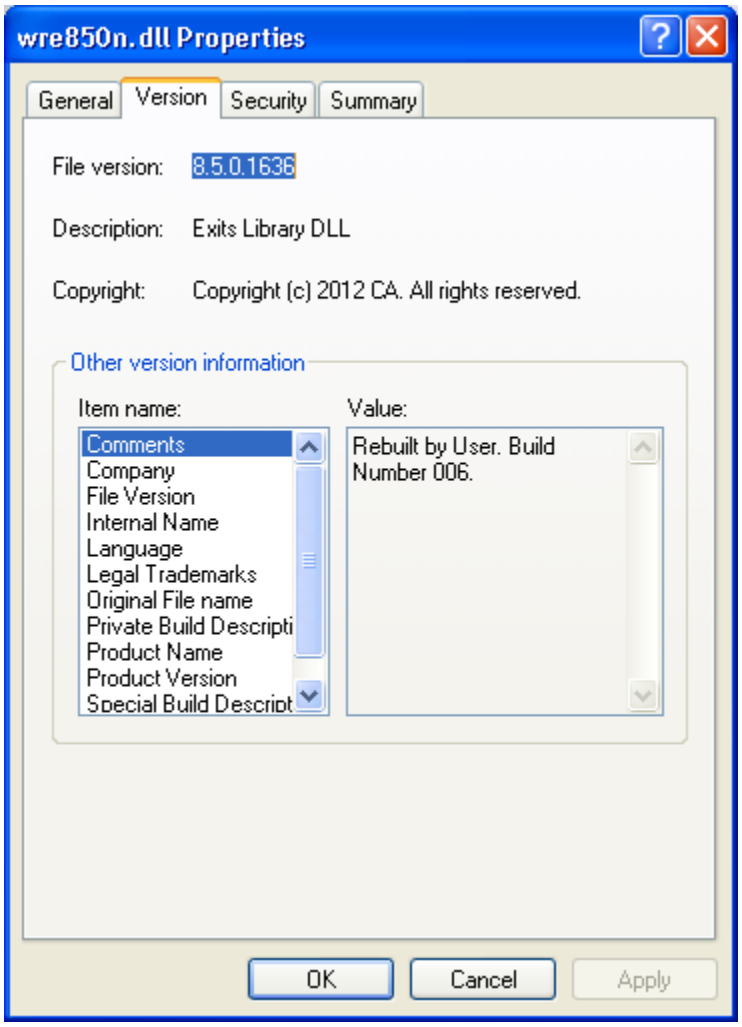
Comments: Will now contain “Rebuilt by User. Build Number XXX”.

Private Build Description: Will now contain timestamp of rebuild

The field “Comments” contains a 3-digit build number that is incremented each time this particular DLL or EXE is rebuilt.

Note: There are no changes required by the customer to utilize this feature.

Once again using the GUI Runtime User Exit DLL wre850n.dll as an example, the Version tab of the Properties dialog looks like this:



From this illustration above, the only noticable difference is the value in the Comment field, as documented above. The value 'Rebuilt by User. Build Number 006’ indicates that this particular DLL was rebuilt 6 times.

The list of Windows IT DLLs and EXEs that will support the modified fields upon a customer rebuild include:

DLL or EXE	Description	Rebuild Procedure
aecdb2**xx**n.dll	Server Db2 Dll	mkdbs.bat
aecodb**xx**n.dll	Server ODBC Dll	mkdbs.bat
aecora**xx**n.dll	Server Oracle Dll	mkdbs.bat
aefsecex.exe	Security Exit	mksecex.bat
aeuexit**xx**n.dll	Server User Exits	mkexits.bat
{dialect}c**xx**0n.dll	All GUI Dialect Global Database Information Container Dlls	mkdialn.bat
{dialect}gui**xx**0n.dll	All GUI Message Table Dlls	mkdialn.bat
stubn.exe	GUI NoDBMS Stub	stubn.mak
stubdb2n.exe	GUI Db2 Stub	stubdb2n.mak
stubodbn.exe	GUI ODBC Stub	stubodbn.mak
stuboran.exe	GUI Oracle Stub	stuboran.mak
ti{dbms}ddl.exe	All DDL loader EXEs	makeddl.bat
wre**xx**0n.dll	GUI Runtime User Exits	mkexitsn.bat
cmicx**xx**n.dll	Client Manager User Exit	ccmexit.nt
ecicx**xx**n.dll	ECI User Exit	ceciexit.nt
mqscx**xx**n.dll	MQ Series Client User Exit	cmqsexit.nt
csuvn**xx**n.dll	CSU Version User Exit	csuglvn.nt
tcpcx**xx**n.dll	TCP User Exit	ctcpexit.nt
tcpux**xx**n.dll	TCP I/O User Exit	inetipux.nt
eciux**xx**n.dll	ECI I/O User Exit	ioeciux.nt
rscux**xx**n.dll	RSC I/O User Exit	iorscux.nt
prex**xx**n.dll	C Proxy User Exit	proxyxit.nt
mqssx**xx**n.dll	MQ Series Server User Exit	smqsexit.nt
iefdirn.dll	Directory Services User Exit	iefdirn.mak
decre**xx**n.dll	CM/CB Decrytion User Exit	makedecr.bat
cide**xx**n.dll	Conversion Instance Data User Exit	makecid.bat
cbmsg{dialect}**xx**n.dll	All CB Message Dlls for Dialects	cbw32\custom.bat

DLL or EXE	Description	Rebuild Procedure
cmmsg{dialect}**xx**n.dll	All CM Message Dlls for Dialects	cmw32\custom.bat
wscx**xx**n.dll	Web Services User Exit	cwsexit.nt

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

Note: User Exits are documented in detail in the *User Exit Reference Guide*.

Index

L

O

P

R

S

T

U

W