CA Gen

UNIX and Linux 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

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 (AEF)
- 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 (AEF) 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. For more information about the AEF, see the chapter Testing and Running Applications.

CA Gen Runtime

The CA Gen Runtime includes libraries the generated application calls to handle low-level and common functionality.

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 information is intended for CA Gen developers using the Implementation Toolset on UNIX or Linux platforms to build and install CA Gen generated applications.

You must know how to configure the development platform to implement generated applications and be familiar with the complier recommended for the platform, Oracle Database, IBM Db2, CA Gen Build Tool, and other technologies required for the specific UNIX or Linux platform.

Chapter 2: Processing Overview

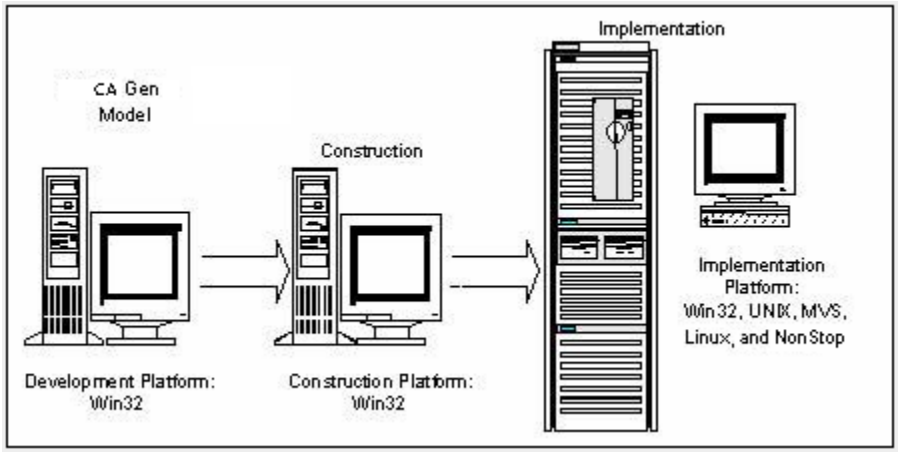
This article includes information about preparing a CA Gen model to use on multiple platforms.

This chapter includes the following topics:

- Target system implementation
- Packaging remote target systems
- The Install Control Module (ICM)
- External action blocks
- CA Gen runtime user exits

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:



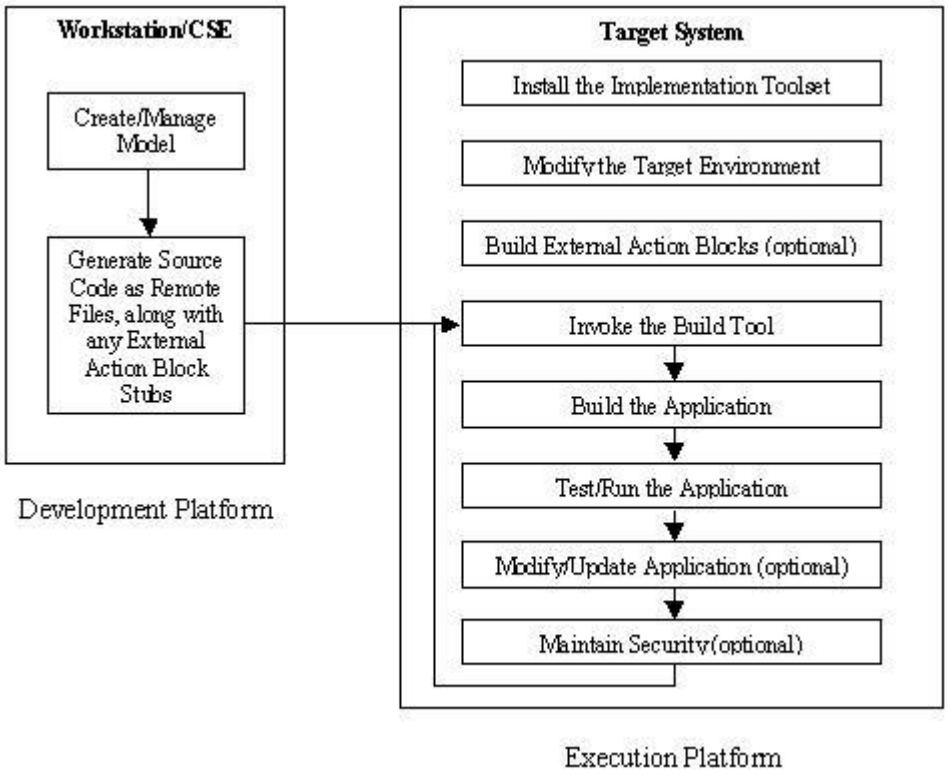
Applications are developed as models using various CA Gen Toolsets on a workstation called the Development Platform. Models are prepared for execution through a process called *construction*, which may occur on the development platform or on another system. The resulting components, called remote files, 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.

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: For more information about installing the Implementation Toolset and the Setup Tool, see *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 shared library. Component Models are packaged into one or more operations libraries.

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 AEF) 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 lets 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 shared 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 on 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 file 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 about transferring remote files to the target system, 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.

Shared versus Archive Library Support

The Build Tool links each UNIX or Linux CA Gen application to a set of runtime libraries delivered with the Implementation Toolset. The Implementation Toolset delivers shared and archive, or static, versions of these runtime libraries. By providing both versions of these libraries, the customer has the flexibility to link either set of libraries to their applications. Linking with shared libraries produces a smaller executable application, with the flexibility to replace libraries without re-linking the application.

The Build Tool token OPT.LIBTYPE determines which set of libraries to link to the customer's application. OPT.LIBTYPE defaults to SHARED to instruct the Build Tool to create a shared library version of the RI Triggers, if applicable, and to link with the shared version of the runtime libraries.

When the customer sets OPT.LIBTYPE to ARCHIVE, the Build Tool creates an archive version of the RI Triggers, if applicable, and links with the archive version of the runtime libraries.

Using the OPT.LIBTYPE token does not impact the set of libraries, if applicable, required from third party software vendors to create CA Gen applications, for example Oracle or MQSeries.

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

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

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.

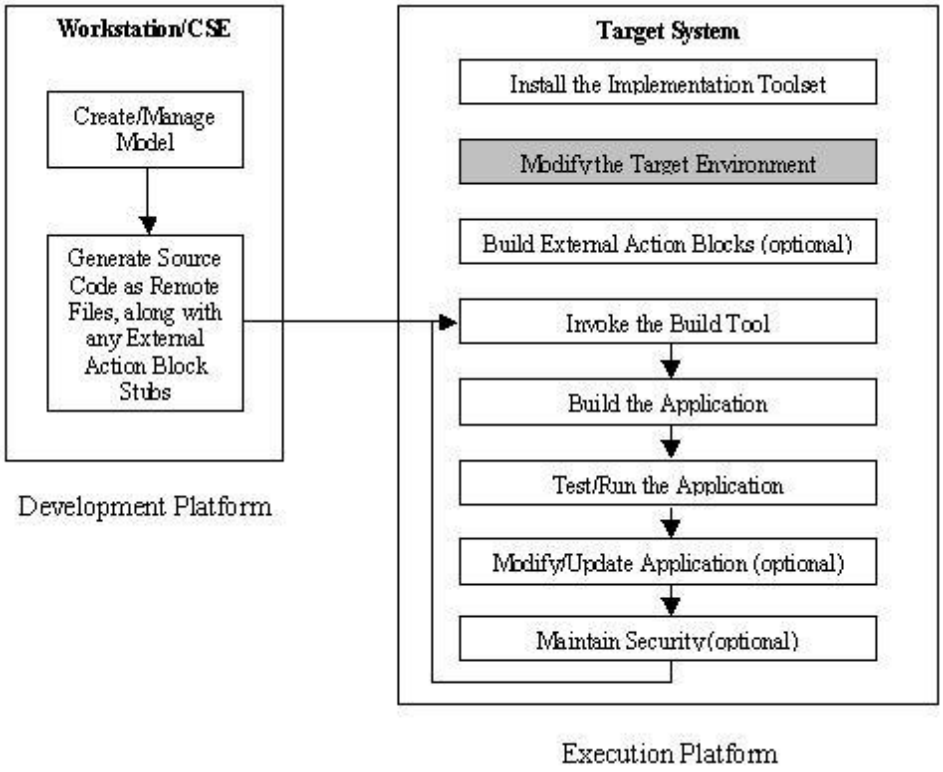
Prerequisites on the Target System

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

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

To facilitate application security:

- The CA Gen Administrator ID and group should own all IT runtime subdirectories and files
- Set up the ID and group before installing the IT.
- Install the IT under the root login.
- Enter the CA Gen Administrator ID and group during installation to ensure all IT Runtime subdirectories and files are owned by the Administrator ID.

More information:

Application Security (see page 59)

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 the 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 AEF.

When set, environment variables retain the new values until the end of the current logon session, or until they are dynamically changed. When a process spawns a new shell, environment variables revert to their original logon values. When exiting the new shell, the variables reset to the values they held before the new shell was spawned.

To display the current environment, use the env command. To view the current value of an environment variable, use the echo command as shown by 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 AEF to reference multiple directories for test and production applications in specific order. Ensure that $IEFH is in AEPATH. You must use the following format:
	`AEPATH=[directory_1;directory_2;...directory_n;$IEFH]`
	Note: On some systems, you must put values that include semicolons in single quotes, as in the following:
	`AEPATH=['directory_1';'directory_2';'...directory_n']`

IEFGXTP	Specifies the directory that contains the codepage translation files. You must use the following format: `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 a CA Gen applications, you must set this variable using the following format: `IEFH=[directory]`
IEFPLATFORM	Name of hardware platform. Valid settings are: ■ HP Itanium ■ IBM AIX ■ SUN Solaris ■ Linux You must use the following format: `IEFPLATFORM=[one of HPia64,IEF_AIX, IER_SOL, or IEF_LINUX]`

Name	Description
LD_LIBRARY_PATH	Specifies the search path for dynamic libraries for Solaris and Linux only. A set of directory paths indicating the dynamic library search order. The path must include $IEFH/lib and DBMS directories, such as $DB2DIR/lib, $ORACLE_HOME/lib. You must use the following format: `LD_LIBRARY_PATH=[directory_1:directory2:…directory_n]`
LIBPATH	Search path for dynamic libraries for AIX only A set of directory paths indicating the dynamic library search order. The path must include $IEFH/lib and DBMS directories, such as $DB2DIR/lib, $ORACLE_HOME/lib. You must use the following format: `LD_LIBRARY_PATH=[directory_1:directory2:…directory_n]`
PATH	Search path for executables. A set of directory paths indicating executable program search order. The path must include $IEFH/bin and '.' for the current directory and DBMS 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 UNIX or Linux 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 UNIX or Linux Profile Table. You must use the following format: `PTOPT=[YES or YY]` ■ Use Profile Table files ■ Compress PT Files
SHLIB_PATH	Search path for dynamic libraries for HP Itanium only. A set of directory paths indicating dynamic library search order. The path must include $IEFH/lib and DBMS directories, such as $DB2DIR/lib or $ORACLE_HOME/lib. You must use the following format: `LD_LIBRARY_PATH=[directory_1:directory2:…directory_n]`

Name	Description
TERM	A character string identifying the type of terminal. You must set this variable before the CA Gen user executes an AEF program on the host system. The p3270keys file in the $IEFH directory must contain this terminal type. For information about the p3270keys file and customizing terminal definitions to meet your needs, see the appendix Keyboard Mapping and Terminal Emulation (see page 67). You must use the following format: `TERM=[terminal_type]`

Optional Variables

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

Name	Description
AEUSER	This variable is used by the profile manager to allow multiple logons with the same logon ID on the same system and application. Set AEUSER to a different value for each new logon that uses the same logon ID. One possible value for AEUSER is the terminal ID. You must use the following format: `AEUSER=username` Also see IEF_USER_ID.
IEF_TERMSYS	Specifies an additional identifier for specific terminal types in the p3270keys file. You must use the following format: `IEF_TERMSYS=[additional_terminal_type_identifier]`
IEF_USER_ID	Sets a unique value for the CA Gen logon ID for the current session. Set this value when accessing the same generated application on one system in multiple sessions. CA Gen generated applications require a unique CA Gen USERID for screen display and CA Gen profile table access. The default CA Gen user ID is the UNIX or Linux logon ID. Since UNIX or Linux lets multiple logons with the same logon ID, use IEF_USER_ID to set a unique value for CA Gen USERID for this session. When AEUSER and IEF_USER_ID are set, action logic uses AEUSER for profiling and IEF_USER_ID for USERID. You must use the following format: `IEF_USER_ID=[unique_login_ID]`

DB2-Specific Variables

The following table describes 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 shared libraries or images built from modified configuration files after reinstalling.

The CA Gen 8.5 IT includes the following configuration files:

- $IEFH/p3270keys
- $IEFH/aeenv
- $IEFH/application.ini
- $IEFH/bin/codepage.ini
- $IEFH/bin/commcfg.ini (excludes Linux)
- $IEFH/bt/scripts/build_unix_lm_c.scr
- $IEFH/bt/scripts/build_unix_ri_c.scr
- $IEFH/bt/scripts/build_unix_ddl.scr

- $IEFH/make/ENVFILE.SAM
- $IEFH/make/RM.sample
- $IEFH/make/UBBCONFIG.SAM
- $IEFH/make/ief_sol.h (Solaris only)
- $IEFH/make/hpia64.h (HP Itanium only)
- $IEFH/make/rs6000.h (AIX only)
- $IEFH/make/ief_linux.h (Linux only)
- $IEFH/src/aefsecex.c
- $IEFH/src/cictuxwsx.c
- $IEFH/src/cictuxx.c
- $IEFH/src/cistuxx.c
- $IEFH/src/cimqclex.c (excludes Linux and HP Itanium)
- $IEFH/src/cimqsvex.c (excludes Linux and HP Itanium)
- $IEFH/src/citcpclx.c (excludes Linux)
- $IEFH/src/ciwsclx.c (excludes Linux)
- $IEFH/src/csuglvn.cxx (excludes Linux)
- $IEFH/src/proxyxit.c (excludes Linux)
- $IEFH/src/tirdb2.sqc (excludes Solaris and HP Itanium)
- $IEFH/src/tirdconn.sqc (excludes Solaris and HP Itanium)
- $IEFH/src/tirora.pc
- $IEFH/src/tiroconn.pc
- $IEFH/src/tirdcryp.c
- $IEFH/src/tirdlct.c
- $IEFH/src/tirdrtl.c
- $IEFH/src/tirelog.c
- $IEFH/src/tirhelp.c
- $IEFH/src/tirmtqb.c
- $IEFH/src/tirncryp.c
- $IEFH/src/tirsecr.c
- $IEFH/src/tirsecv.c
- $IEFH/src/tirserrx.c

- $IEFH/src/tirsysid.c
- $IEFH/src/tirterma.c
- $IEFH/src/tirupdb.c
- $IEFH/src/tiruppr.c
- $IEFH/src/tirurtl.c
- $IEFH/src/tirusrid.c
- $IEFH/src/tirxlat.c
- $IEFH/src/tiryyx.c

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.

More information:

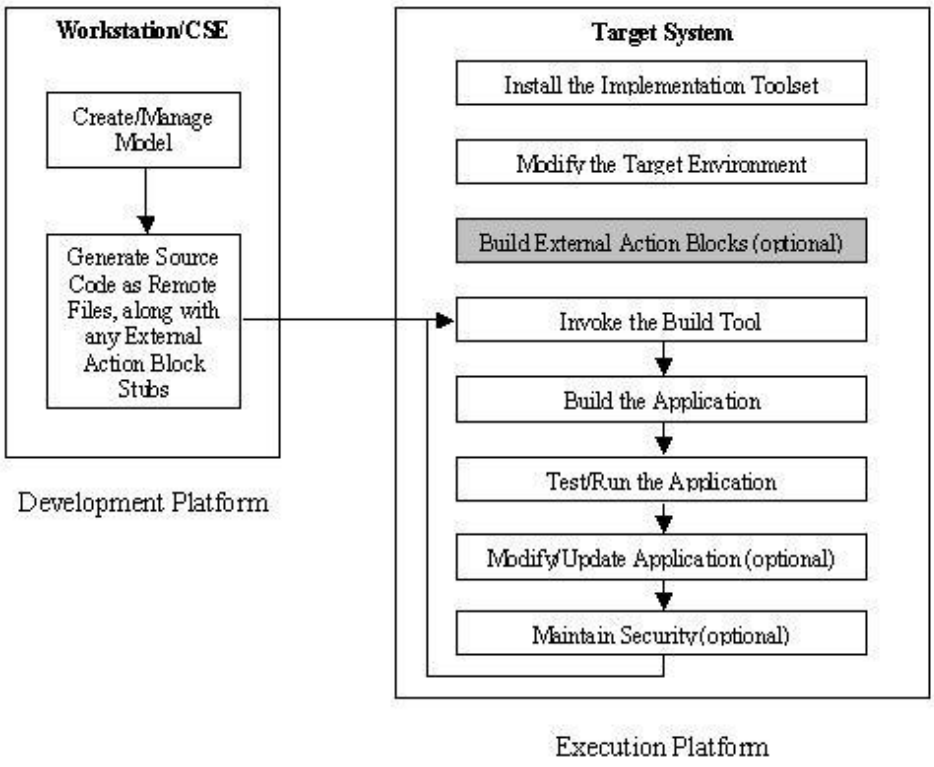
User Exits in UNIX and Linux (see page 65)

Chapter 4: External Action Blocks

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. For a list of conditions that must be satisfied on the development platform prior to implementing EABs on your target system, see the chapter Prerequisite Implementation Tasks (see page 19).

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 a code is generated. The CA Gen software cannot generate actual code for EABs, but the software creates 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 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:

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 with leading and 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 the 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

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

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, w_oa

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

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*.

Test an External Action Block

After the load module that uses the EAB builds successfully, you are ready to test the load module.

More information:

Testing and Running Applications on UNIX and Linux (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.

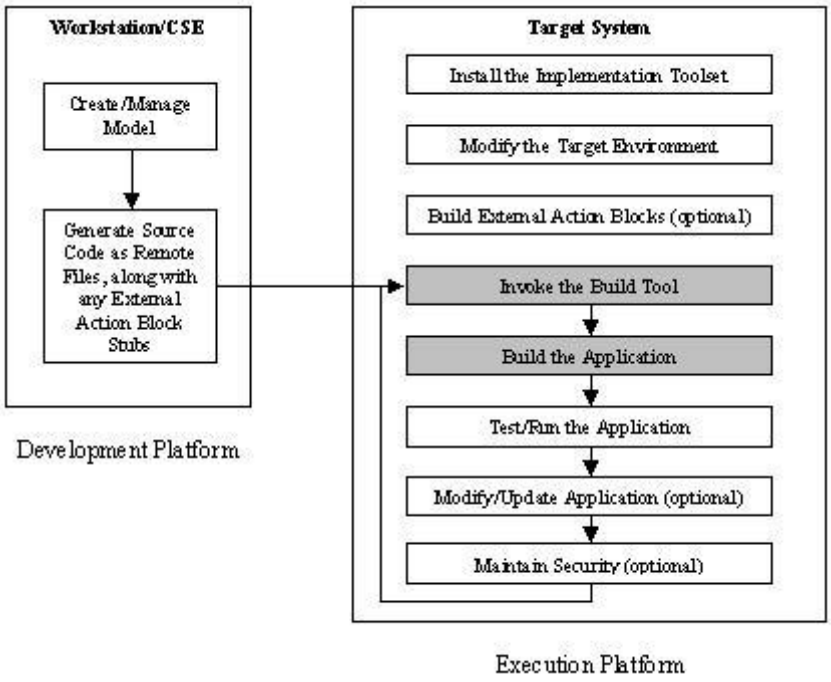
If the EAB is a private operation in the operations library, the user must ensure that no function in the consuming model uses the same name as the private EAB function. The Build Tool cannot guarantee the outcome of an application that has multiple declarations of the same function.

Chapter 5: Processing Remote Files in UNIX and Linux

Before processing remote 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 remote files and other IT tasks:



Invoke the Build Tool

After generating the remote files on the development platform, they are:

- Transferred to the target system, through the Build Tool or outside of CA Gen
- 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*.

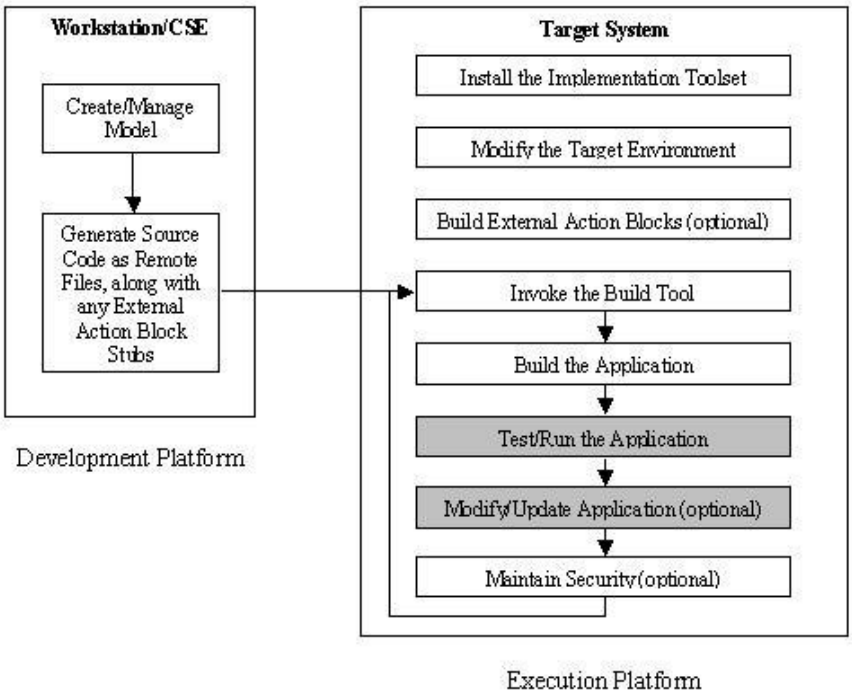
Recompile and Relink Application Considerations

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 on UNIX and Linux

Although an application is ready for execution after processing the remote 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.



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 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 AEF as the interface, or a set of C Servers are built, the executables and the triggers library are copied to the models' /inqload subdirectory. When a C block mode command line application is built, the executables and the triggers library are copied to the models' /bin subdirectory. Separating the locations of the executables allows both block mode application styles to co-exist for one model. In all cases, a local version of the AEENV file is updated within the models' source directory.

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 on RESET and RESTART, see the *Toolset Help*.

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 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.

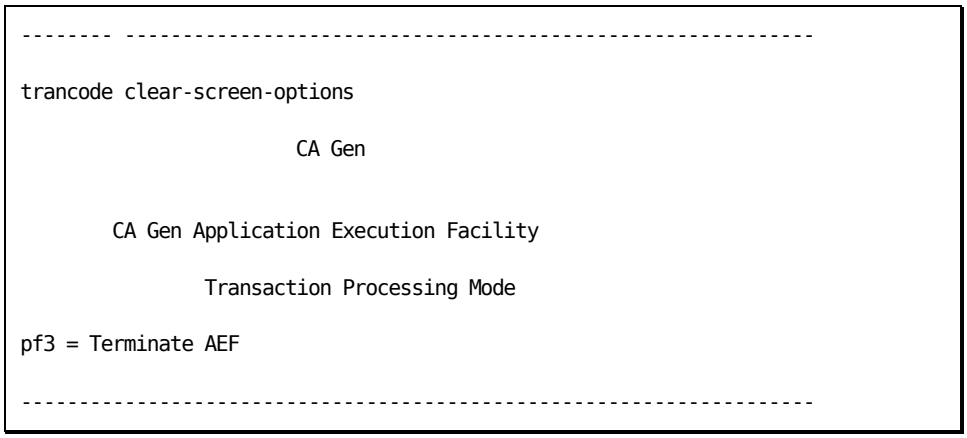
Setting Environment Variables

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 stored in the models' /bin subdirectory or /inqload subdirectory. These directories specify the search order for routing tables and load modules. As a requirement, AEPATH must contain the suffix $IEFH to locate the p3270keys file. AEPATH supports test and production directories. Example: setenv AEPATH <model directory>":"$IEFH 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: setenv IEFGXTP /home/ptauto/it/gen/runtime/translat
IEFH	Defines the home directory for the Implementation Toolset that contains the CA Gen libraries and binaries, and identifies the location of the p3270keys used by the AEF and the AEFAD. **Note:** Set IEFH when installing the Implementation Toolset Example: setenv IEFH /home/ptauto/it/gen/runtime
LD_LIBRARY_PATH	Solaris only. Defines the shared library path(s) for Solaris systems and enables applications to locate all shared libraries when the application executes. Include the DBMS library path, and the CA Gen runtime library path, $IEFH/lib.
LIBPATH	AIX only. Defines the shared library path(s) for AIX systems and enables applications to locate all shared libraries when the application executes. Include the DBMS library path, and the CA Gen runtime library path $IEFH/lib.
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.
SHLIB_PATH	HP Itanium only. Defines the shared library path(s) for HP Itanium systems and enables applications to locate shared libraries when application executes. Include the DBMS library path, and the CA Gen runtime library path $IEFH/lib.

Application Execution Facility

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



Note: To execute applications through the AEF, 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 AEF from a terminal session, using the following command syntax:

```
aef [options]
```

The following table defines the AEF 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.

Option	Description
-t [number]	Value between 0 and 31 that activates an internal AEF program trace. 15 is the most useful level. This option creates a file in the local directory containing the trace data. The filename is lg-<procname>-<procid>.log where <procname>is the name of the program running, the AEF or the load module, and <procid> is the process number of the program running. Example: lg-aef-163574.log
-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.
-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 using the AEF. Server or command line applications are not invoked through the AEF.

Function key assignments for the AEF may differ for each terminal type. The AEF uses a keyboard mapping file, p3270 keys, to assign values for each function key and special purpose key available for generated applications on the target system. The file is organized by terminal type and contains a set of entries for each terminal type usable with your target system. For more information about the p3270 keys file, see the appendix Keyboard Mapping and Terminal Emulation in this guide.

The AEF 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 AEF to locate the aeenv files. The AEF reads the trancode entered, and invokes the associated load module. When the load module terminates, control returns to the AEF.

Trace Options

Use trace options only under the direction of Technical Support to locate problems. The log file containing the trace data is created in the local directory. The filename is named lg-<procname>-<procid>.log where <procname> name of the program running, the AEF or the load module, and <procid> is the process number of the program running. For example, lg-aef-163574.log. A trace value determines the level of information logged to the file. Trace values range from 0 to 31, where 0 provides the least information and 31 provides the most information. The exact amount of information stored by each value may vary from release to release.

Increasing the trace value also increases the file size. A trace value of 31 may consume a large amount of disk space that can significantly slow processing. Use trace options carefully in production systems.

Key Mapping

When the application starts through the AEF, keyboard mapping translates terminal keystrokes to the function key structure the application uses. Appropriate key mapping should be established before executing applications. For more information about key mapping, see the appendix Keyboard Mapping and Terminal Emulation in this guide.

More information:

Keyboard Mapping and Terminal Emulation (see page 67)

Application Execution Facility Record and Playback Feature

The AEF 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 AEF 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 AEF 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 AEF 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 AEF record and playback feature to:

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

Limitations

The AEF record and playback feature includes the following limitations:

- Always delete the recording file to start fresh because AEF 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 AEF for Record/Playback

To start recording, enter the following command:

```
aef -s [filename]
```

To initiate script playback, enter the following command:

```
aef -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 CA Gen model elements used to build the application, such as the 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

To use the Diagram Trace Utility

1. Build the test application generated with trace.
2. Invoke the Diagram Trace Utility on any Windows 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.

   Note: For more information about using the application.ini file and the trace environment variables, see the appendix The Application.ini File (see page 91).
4. Set the AEPATH environment variable when invoking the application through the AEF.

Access and Use the Diagram Trace Utility

You can use the Diagram Trace Utility to test all generated C applications on UNIX and Linux systems. 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 to Obtain Trancode
Distributed Processing Server	AEFAD or Websphere MQ or Tuxedo	From the AEENV file
Block mode Command line	Command Line	As a parameter when invoked, from the AEENV file during execution
Block mode Interactive	AEF	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 this 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 that client workstation.

To use the client's Diagram Trace Utility

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.

Note: To provide the server with the client's host and port information, these values are added to the Common Format Buffer (CFB). A CFB is used to exchange view data between the client and server. If the addition of the host and port values exceeds the allowable size of the CFB, the host and port values are not added to the CFB and the statement "*Not enough space in CFB to pass Trace flags to server*" is logged to the client's trace file. The end result is that no override takes place, and the values of the host and the port will be extracted from the server's application.ini file.

Overriding Application.ini for Block Mode Testing

In certain situations, it may be necessary for multiple users to test a Block Mode application. For Block Mode applications, you can override the contents of the Application.ini file with each invocation of the Block Mode application by setting four environment variables before invoking the Block Mode application.

These environment variables are:

- TRACE_OVERRIDE

  When the TRACE_OVERRIDE environment variable is set to 1, the trace environment variables contained in the Application.ini file are overridden by the environment variables set in the command shell.

- The Application.ini file contains the following environment variables:
  - TRACE_ENABLE
  - TRACE_HOST
  - TRACE_PORT

More information:

The Application.ini File (see page 91)

Regenerate 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 AEF. The AEF 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

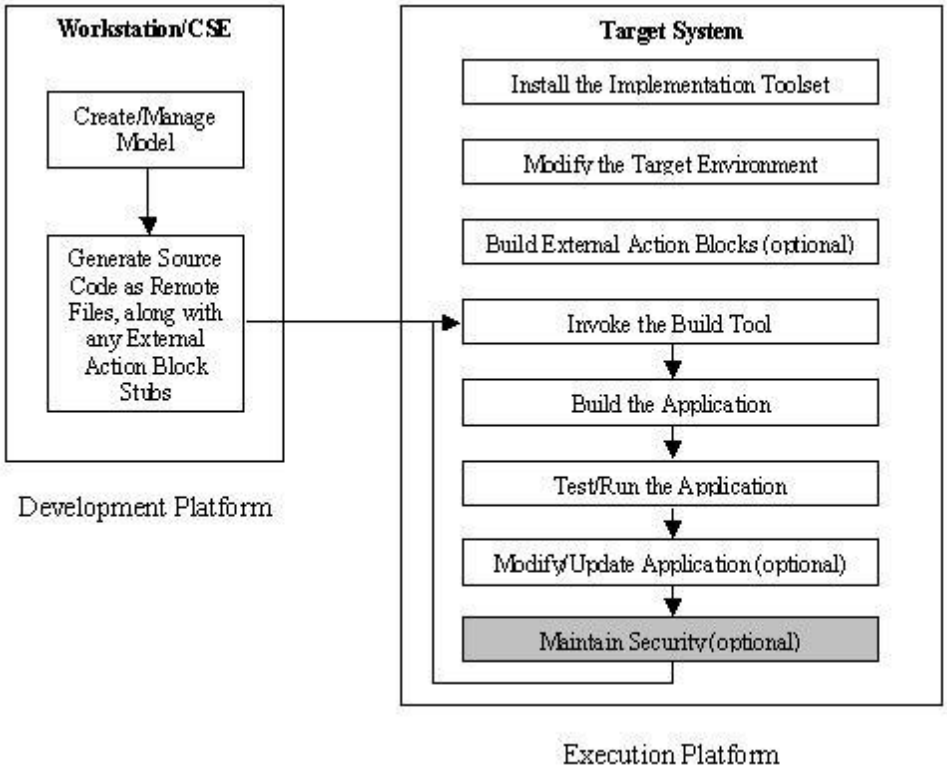
To test changes to a production application, isolate the test environment from the production environment by installing the remote files in a different location and generating a test database using a different name than the production database name. This ensures that the remote files generated for test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

Chapter 7: Application Security

This chapter provides information on how to implement security to the application.

Implementation Sequence

After an application is in production, it is important to protect its database. The following illustration shows the relationship of application security to other IT tasks:



Background

A basic security problem occurs because the AEENV file is an ASCII text file that contains DBMS ID and password information. To run a transaction, a user must have permission to read the AEENV file that associates trancodes, load modules, and DBMS connect information. When users have access to this file, they can also browse the file, read the DBMS ID and password, and connect to the database using generic utilities to query and manipulate the database, unrestricted by the generated application's data integrity and security processing.

One solution to this problem uses a standard UNIX and Linux file access capability that grants a user the file access privileges of another user only while running a program the other user owns. Specific users can use CA Gen applications while preserving database ID and password information security.

CA Gen Administrator ID

The first step to application security is to create a UNIX or Linux CA Gen Administrator ID and a special UNIX or Linux user group for that ID. This UNIX or Linux ID and group is the main control point for access to applications that require security. Only disclose the Administrator ID and password to users that perform administrative duties on production applications. The root user enters the ID and group during Gen installation.

AEENV File Access

Identify the directory that stores production applications. In this directory, create an AEENV file to store trancode and load module information for production transactions and create a subdirectory called /inqload that stores the p3270 production application load modules. Only the CA Gen Administrator ID should have read and write access to these files and directories.

Transactional Applications

Transactional applications are run using the AEF, in the $IEFH/bin directory. Change to the $IEFH/bin directory and modify the permissions of the executable program, called aef, using the following command:

```
chmod 4711 aef
```

The permissions should read:

```
-rws--x--x aef
```

The s in the permissions field indicates that a user executing this program inherits the privileges of the ID that owns the executable. Anyone using the AEF to run an application has read access to the production AEENV file and the load modules while running the AEF. When the application needs database access, the application can read the AEENV file. The user can not browse the files.

Command Line Applications

Command applications, with and without screens, are fully linked executable programs that do not require the AEF to run.

Put the executables for command applications in a secure directory owned by the CA Gen Administrator ID, with the following permissions:

```
-rws--x--x
```

The programs are executable by anyone and the s flag in the permission field indicates that the user's effective ID becomes that of the CA Gen Administrator, giving the user access to the proper AEENV file.

Multiple UNIX or Linux Groups

Using multiple UNIX or Linux groups to control access to applications can provide additional security when required. This section includes examples of other ways to control application access.

Sample Procedure for Controlling Application Access Using Multiple UNIX or Linux Groups

The following is a sample procedure for controlling application access using multiple UNIX or Linux groups:

1. Create two UNIX or Linux groups:
   - mktg—The marketing group
   - acct—The accounting group
2. Create a unique CA Gen Administrator for each group.
3. Assign each UNIX or Linux ID to one of the groups.
4. Create a directory for each group's applications. The CA Gen Administrator owns the directory for their group.

   Copy the p3270keys file into the group's application directory.
5. Create a /bin directory named in each group's application directory.
6. Copy the AEF executable, called aef, into the /bin directory and change the permission to:

   ```
   -rws--x--
   ```
7. Change the owner to the group's CA Gen Administrator, and the owning group to the group name.

8. Create an /inqload subdirectory and an AEENV file in the group's directory.
9. Install the group's applications in the /inqload subdirectory
10. Each group member should access their new environment and executables using the correct environment variable settings.

Security Using the dbconnct User Exit

The default method for CA Gen to acquire DBMS connection information for block mode applications is for the AEF to locate the trancode in the AEENV file. Connection information includes the username, password, and database name. Each person that executes the application must have read access to the AEENV file that contains the connection information.

To enhance security, the following user exits handle the specific DBMS connection logic:

DBMS	User Exit
Oracle	$IEFH/src/tiroconn.pc
DB2	$IEFH/src/tirdconn.sqc

By default, these modules read the database connect information from the AEENV file and use the information in the database connect statement. To modify the default for obtaining database connect information, follow these steps:

1. Copy the module you want to modify and edit the appropriate user exit.
2. Add logic to populate the variables required for the database connections.

Database	Description	Host Variable	Declared Type
Oracle	user ID	uid	VARCHAR(32)
Oracle	password	pwd	VARCHAR(32)
DB2	user ID	uid	char(9)
DB2	password	pwd	char(9)
DB2	database name	dbname	char(9)

We recommend leaving the call to dbid() unchanged, and adding logic immediately before the database connect statements to populate the appropriate variables. Ensure that you add all code that the DBMS requires. For example, verify arr and len elements are populated correctly for VARCHAR. We also recommend that all AEENV files contain character strings as place holders for the database connection information. These character strings do not have to contain valid connection information.

After modifying the user exit, use the build command $IEFH/make/mkdbs to precompile, compile, and archive the database connection module for linking all applications to use the new logic.

Recreate the shared libraries for the runtime from the updated archive libraries. The simplest modification is to add strcpy() statements to the code to set the host variables. This hard codes the values for the database connection information into the user exit and requires taking appropriate file security measures. For greater security, add a call to an encryption routine.

Chapter 8: User Exits in UNIX and Linux

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 $IEFH/src directory.

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

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 C 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 through the script $IEFH/make/mkexits. You must relink all statically linked applications that use the recompiled user exit before the change becomes effective. You do not have to relink applications built using shared runtime libraries.

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 the *Release Notes*.

The following user exits are available in C:

Name	Description
TIRYYX	Date User Exit that defines the century for a two-digit year. Refer to the source module for detailed information.
TIRDCRYP	Decrypt User Exit

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

Note: The database user exits DBCONNCT, DBCOMMIT and DBDISCNT are rebuilt into individual static libraries and individual shared libraries using the script $IEFH/make/mkdbs. For information about these exits, see the *User Exit Reference Guide*.

More information:

Rebuilding DBMS Shared Libraries (see page 79)

Chapter 9: Keyboard Mapping and Terminal Emulation

This appendix details the Implementation Toolset's (IT) keyboard mapping capabilities relative to the UNIX or Linux operating system, and setup instructions for terminals that contain built-in setup capabilities.

Note: Since there are so many terminal types available to use with UNIX or Linux operating systems, these configuration instructions may not apply to all of them. For additional information, see the manual for your computer hardware, the software manual for TCP/IP, or your communications software.

Up to 24 special key combinations are available for the generated applications that run on the target system using the AEF environment. These key combinations, the majority of which are function keys, provide IT and target system application users an efficient and easy way to perform special or redundant screen operations, such as canceling an operation, invoking application help, or requesting a list of selectable items.

However, different terminal types may have function key capabilities that fail to exactly match those required by the IT. Keyboard mapping provides the capability of equating CA Gen function key requirements to individual keys or equivalent sets of keystrokes from a particular terminal type. Using this feature, most terminals types can be easily adapted to optimally use the IT and generated applications capabilities.

The keyboard mapping file is named p3270keys and is in the $IEFH directory. When the AEF fails to locate the p3270keys file, or the file does not include the required terminal definition, the AEF searches for terminal definition information in the UNIX or Linux-supplied terminal information database, TERMINFO.

TERMINFO is a compiled executable datafile that provides quick access to terminal definitions. For additional information about the TERMINFO database, see your UNIX or Linux system documentation.

Keyboard Mapping File

The p3270keys keyboard mapping file translates a set of keystrokes generated from the terminal of a target system into the function key structure the IT and the generated applications use.

This translation is done when the AEF obtains ASCII terminal input and output string definitions from the p3270keys file that contains input key-to-function definitions to associate a character string received from the keyboard with a particular 3270 control or function key. The AEF also uses the p3270keys file definitions to determine the character string to output to the terminal to execute a particular screen function.

The IT includes a standard p3270keys definition file with all the elements required to describe a particular terminal environment operation, such as function key definitions, color commands, and screen display descriptions. The IT install places the p3270keys file in the $IEFH directory as a text file that can be edited to add new key mapping or changed to reflect different functions for existing terminal types.

Keyboard Mapping File Terminal Definitions

Each set of terminal definitions begins with the string term= and continues with input and output string definitions. The format is shown in the following example and defined in the following table.

```
term= [IEF_TERMSYS value]
term= [alternative IEF_TERMSYS value]
mnemonic name or phrase= definition string [ ]
@endterm
```

Element	Description
term=	Identifies the terminal type specified in the environment and marks the start of a set of definitions for a terminal. If a terminal type has more than one name, include an additional term line for each name. If there are multiple "term=" terminal definition blocks for different platforms, an associated platform value is associated with the "term=" keyword delimited by [], determined through the environment variable IEF_TERMSYS. **Note:** The name specified by the user with the TERM environment variable must exactly match the term= name in this file to use the definition.
mnemonic name or phrase=	Identifies the corresponding AEF key or function.
definition string [ ]	Identifies the input key string or output function string.
@endterm	Ends the set of key definitions for a particular terminal.

Each input key string or output function string is on a separate line with the mnemonic name or phrase (symbol) and the definition string. A space or end of line terminates the definition string. For output string definitions, the remainder of the line is ignored and may be used for comments. For input key string definitions, the key, or keys, generating the input string is enclosed in brackets. AEF displays the key when a user requests HELP using Esc h.

The following is an example of a keyboard mapping file:

```
term=ps2  [HP9000]
backtab=^@
lines=25
enter=\EOM [NumPad Enter]
crgret=^M [Enter]
delete=\EOn
lns=\EOp
down=\EOB
upkey=\EOA
left=\EOD
right=\EOC
home=\EOw
eraseof=\Ef [Esc f]
eraseIn=\Es [Esc s]
pa1=\EOE [Alt F5]
pa2=\EOF [Alt F6]
pa3=\EOG [Alt F7]
pf1=\EOP
pf2=\EOQ
pf3=\EOR
pf4=\EOS
pf5=\EOT
pf6=\E[17~
pf7=\E[18~
pf8=\E[19~
pf9=\E[20~
pf10=\E[21~
pf11=\E[23~ [shift F1]
pf12=\E[24~ [shift F2]
pf13=\E[25~ [shift F3]
pf14=\E[26~ [shift F4]
pf15=\E[28~ [shift F5]
pf16=\E[29~ [shift F6]
pf17=\E[31~ [shift F7]
pf18=\E[32~ [shift F8]
pf19=\E[33~ [shift F9]
pf20=\E[34~ [shift F10]
pf21=\E[OK [Alt F1]
pf22=\E[OL [Alt F2]
pf23=\E[OO [Alt F3]
pf24=\EON [Alt F4]
```

```
blink=\E[1;5m
normal=\E[0;1m
reset=\E[0;1m
reverse=\E[0;7m
underscore=\E[0;4m
blue normal=\E[0;1;36m
red normal=\E[0;1;31m
magenta normal=\E[0;1;35m
green normal=\E[0;1;32m
white normal=\E[0;1;37m
cyan normal=\E[0;1;36m
yellow normal=\E[0;1;33m
white normal=\E[0;1;37m
blue reverse=\E[7;1;34m
red reverse=\E[7;1;31m
magenta reverse=\E[7;1;35m
green reverse=\E[7;1;32m
white reverse=\E[7;1;37m
cyan reverse=\E[7;1;36m
yellow reverse=\E[7;1;33m
white reverse=\E[7;1;37m
blue blink=\E[5;1;34m
red blink=\E[5;1;31m
magenta blink=\E[5;1;35m
green blink=\E[5;1;32m
white blink=\E[5;1;37m
cyan blink=\E[5;1;36m
yellow blink=\E[5;1;33m
@endterm
```

Input Strings for the Keyboard Mapping File

Note: Some terminals lack the ability to send different characters for the Return and numeric keypad Enter keys. In those cases, select an unused key, determine what sequence of characters it currently sends, and modify the p3270keys file for that terminal to recognize the unused key as an Enter or Return key.

The following table defines the keywords that identify AEF input keys and default strings:

p3270keys Name	Termcap ID	Default	Function
	\Eh		Display keyboard mapping help screen
	^c		Exit AEF
backtab	bt	\El	Move cursor to prior unprotected field

p3270keys Name	Termcap ID	Default	Function
chghat		\Ed	Switch attribute display character
clrkey		\Em	Clear screen
codes		\Et	Display input keys/codes mapped to 3270 key
crgret		^m	Move cursor to first unprotected field on next line
delete	kD	\EW	Delete character at cursor
down	kd		Move the cursor down one line
enter		^j	Enter request to application
eraseof			Erase to the end of current field
home	kh		Move cursor to the first unprotected field
ins	kI		Toggle insert mode
left	kl	^h	Move the cursor left one character
pa1			Input PA1 key to the application
pa2			Input PA2 key to the application
pa3			Input PA3 key to the application
pf1	k1		Input PF1 key to the application
pf2	k2		Input PF2 key to the application
pf3	k3		Input PF3 key to the application
pf4	k4		Input PF4 key to the application
pf5	k5		Input PF5 key to the application
pf6	k6		Input PF6 key to the application
pf7	k7		Input PF7 key to the application
pf8	k8		Input PF8 key to the application
pf9	k9		Input PF9 key to the application
pf10	k0		Input PF10 key to the application
pf11	kB		Input PF11 key to the application
pf12	kC		Input PF12 key to the application
pf13			Input PF13 key to the application
pf14			Input PF14 key to the application
pf15			Input PF15 key to the application
pf16			Input PF16 key to the application

p3270keys Name	Termcap ID	Default	Function
pf17			Input PF17 key to the application
pf18			Input PF18 key to the application
pf19			Input PF19 key to the application
pf20			Input PF20 key to the application
pf21			Input PF21 key to the application
pf22			Input PF22 key to the application
pf23			Input PF23 key to the application
pf24			Input PF24 key to the application
resetkey		\E\E	Reset lock and clear insert mode
right	kr		Move the cursor to the right one character
tab	ta		Move the cursor to the next unprotected field
up	ku		Move cursor up one line

Developing Input String Definitions for a Terminal Type

Use the following procedure to develop input string definitions for a terminal type:

1. Invoke the AEF.
2. Press Escape
3. Press the t key to turn the input key codes display on.
4. Press a key that corresponds to a function named in the p3270keys file.

   The string that defines that key displays, beginning in column 54 of the 25th status line. If the AEF understands the key, the key name displays.
5. In the p3270keys file, type the key name= string.

   For example, pressing the F1 key on an ANSI terminal displays \E[M followed by pf1 because the term=ansi section of the p3270keys file has the following line:

   ```
   pf1=\E[M [F1]
   ```

   If the F1 key is undefined, only the \E[M string displays.
6. Press Escape
7. Press the t key to turn the input key codes display off.

Output Strings for the Keyboard Mapping file

Terminfo definitions include the ASCII strings for output functions such as clear screen and cursor motion and are listed in the following table. The file does not include standard codes for defining colors and highlighting. The p3270keys file contains the character string for colors, highlights, or both.

p3270keys Name	Termcap ID	Function
blink	mb	Start blink
ce		Clear screen to end of line
cl		Clear screen
cm		Cursor motion
md		Start bold
reset	me	Ends a previous screen attribute or mode. For example stops bold, blink, or reverse fields.
reverse	so	Start stand out
underscore	us	Start underscore

The following table lists the key words to identify colors and highlights:

Note: When using monochrome terminals with applications that set color or other special attributes, modify the p3270keys file entry for that particular terminal to send NULL (00) characters instead of escape sequences.

Colors	Highlights
Blue	Blink
Cyan	Normal
Green	Reverse
Magenta	Underscore
Red	
White	
Yellow	

The following tables list the function and special keys available to use with generated applications. These tables also provide room to enter the specific keystrokes that correspond to the function, cursor movement, and special keys that correspond to your particular terminal. Print the tables and complete the tables for each type of terminal used with the IT or a generated application.

Terminal-Specific Function Key Definitions

Terminal-specific function key definitions are shown in the following table.

Key	Terminal Definition	Key	Terminal Definition
F1		F13	
F2		F14	
F3		F15	
F4		F16	
F5		F17	
F6		F18	
F7		F19	
F8		F20	
F9		F21	
F10		F22	
F11		F23	
F12		F24	

Terminal-Specific Cursor-Movement Key Definitions

Terminal-specific cursor movement key definitions are shown in the following table.

Key	Terminal Definition	Key	Terminal Definition
Backspace		Insert	
Back tab		Left	
Clear screen		Reset	
Delete		Return	
Delete to end of line		Right	
Down		Tab	
Enter		Up	
Home			

TERMINFO Database

When the AEF fails to locate the p3270keys file, or the file does not include the required terminal definition, the AEF searches for terminal definition information in the UNIX or Linux-supplied terminal information database, TERMINFO.

TERMINFO is a compiled executable datafile that provides quick access to terminal definitions. For additional information about the TERMINFO database, see your UNIX or Linux system documentation.

When the AEF fails to locate a definition in the p3270keys file and TERMINFO, it substitutes a default definition, if it is available. When it cannot define an essential key, it terminates with an error message that identifies the undefined key or keys.

When an entire string specified for one key matches all or part of the string for a different key, the AEF continues processing and displays a message warning that the particular keys may not work properly.

Terminals with Built-in Setup

Some terminals have built-in setup capabilities the user must configure. After setup and before invoking the AEF, ensure the setup is correct and that the p3270keys entry matches the setup.

AEF P3270 Emulator

With the UNIX or Linux attached terminal, the AEF P3270 emulator emulates an IBM 3279 terminal without graphics capabilities. It maintains a 24 x 80 screen image, and a 25th status line. If a UNIX or Linux terminal does not have a 25th line available on the display, the AEF might use line 24 to display status information.

Activation

The command aef executes the AEF P3270 emulator. When the AEF program starts, it presents a 3270-formatted screen to the user and the 3270 emulator is active. The user enters the appropriate program name or transaction code to begin an interactive or transaction application.

Error Conditions

If a failure occurs when the AEF program starts, it indicates the P3270 terminal emulation is malfunctioning and may have terminated.

The AEF issues the following message when it detects a missing p3270keys file or definition:

```
Term setup file is missing/wrong!
```

This message may occur because the AEF failed to locate the p3270keys file, based on the IEFH environment variable, or because the terminal type in the TERM environment variable is undefined in the p3270keys file and the TERMINFO database.

Keyboard Lock

When the user attempts to enter data incorrectly, the keyboard locks and the 25th status line lists an X and a symbol identifying the reason the keyboard locked.

To unlock the keyboard, press the CA Gen reset key.

Status Line

The AEF P3270 emulator uses the 25th line on the terminal screen to hold encoded operator information. Operator information can begin in different columns. On 24-line terminals, the status information displays on line 24 alternately with the data normally on line 24.

The following table summarizes the operator information that appears on the 25th status line:

Column	Symbol	Description
1	[4]	AEF P3270 emulator is ready and the keyboard task initiated.
5	B	AEF P3270 emulator is connected to a program.
7	host	AEF P3270 emulator is operating on the host system.
16	X	Input is inhibited and the symbol in column 18 indicates the reason.
16	?+	The last input was not accepted. This usually occurs when the user attempts to input data while input is inhibited. Press Escape twice to clear.

Column	Symbol	Description
18	WAIT	Input is inhibited to allow time for application execution.
18	OP>NUM	The user attempted to enter non-numeric data in a numeric field. Entered character is not 0 - 9 or a dash (-). Press Escape twice to clear.
18	<OP>	The user attempted to modify a protected field or attribute byte. Press Escape twice to clear the keyboard lock and move the cursor to another location.
18	OP>	The user attempted to enter too much data into the field. Press Escape twice to clear.
36	NUM	The cursor is in a numeric field. Only 0 - 9 or a dash (-) are accepted.
36	PRO	The cursor is in a protected field. No modification is allowed.
36	ATR	The cursor is on an attribute byte. No modification is allowed.
36	KBD	User has started modification to screen from keyboard. Input is not inhibited when this appears.
52	^	Keyboard is in insert mode, inserting characters at the cursor position. The INS key toggles insert mode on and off.
54	codes	Key codes for last keys typed and the 3270 mapped key. Turned on or off with Esc t.

Termination

Request normal termination of an AEF program using the AEF Clear key, often the F3 function key. The AEF exits without issuing a message.

^C also terminates the AEF and causes the AEF process to stop. Because ^C may produce unpredictable results, avoid using it.

Chapter 10: Rebuilding DBMS Shared Libraries

CA Gen supplies a set of DBMS shared libraries that include default connect, disconnect, commit, rollback, and error behavior for the set of supported DBMSs.

Customers who have DBMSs requirements that differ from those supported, can rebuild the appropriate shared library using the supplied build procedure $IEFH/make/mkdbs. This appendix describes rebuilding the shared libraries.

DBMS Shared Libraries

Rebuild DBMS shared libraries using the build procedure $IEFH/make/mkdbs. The following table lists the set of shared libraries that can be rebuilt:

File Name	Supports
libae_db2.{ext}	DB2
libae_oracle.{ext}	Oracle

Note: {ext} refers to the shared library file extension. This differs for some UNIX or Linux operating systems. For HP Itanium, Solaris and Linux it is so. For AIX, it is so.a.

DBMS Shared Library Rebuild

Use the file $IEFH/make/mkdbs to rebuild the appropriate shared library when the DBMS version is different than specified in the *Technical Requirements* document. Mkdbs is invoked as follows:

```
mkdbs
```

It prompts for a DBMS to rebuild:

```
Which Database Shared Library do you want to rebuild?
O – Oracle
D – DB2
enter letter >
```

Note: The PATH, LIBPATH, and SHLIB_PATH environment variables need to contain proper entries for the appropriate DBMS chosen.

Altering the DBMS Version

If the customer wants to alter the DBMS version they use with the block mode and Server Runtimes, they must make the following hand-edits in the UNIX or Linux Runtime environment:

1. Update the platform header file: $IEFH/make/<platform>.h:

   Use the <platform>.h file when building an application. Modify or confirm the following script variables when altering the DBMS version:

   - XALIBS - Database library for XA usage (Oracle only)
   - LIBS - Database library list
   - PCC - Precompiler location
   - PCCINC - Precompiler includes
   - DBMSINC - Database includes

2. Run $IEFH/make/mkdbs:
3. The mkdbs procedure updates the database library and rebuilds the DDL loader for the DBMS.

Database Loader Rebuild

When mkdbs executes, it 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 shared libraries and executables are built with the same DBMS version.

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

Filename	Supports
TIDB2DDL.EXE	DB2
TIORADDL.EXE	Oracle

Chapter 11: Troubleshooting in UNIX and Linux IT

This article lists common problems that might occur when using the Implementation Toolset (IT) in UNIX and Linux. It is divided into these two sections:

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

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

AEF 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 AEF starts.	Ensure the TERM environment variable is properly defined in $IEFH/p3270keys
`Term setup file missing/wrong.` message when the AEF starts.	■ Ensure the IEFH is correctly set. ■ Ensure $IEFH/p3270keys properly defines the TERM environment variable. ■ Ensure you have read privileges for the p3270keys file.
`Command not found.` Message when the AEF starts	■ Ensure PATH includes $IEFH/bin ■ Ensure you have execute privileges for $IEFH/bin/aef.

DBMS Troubleshooting

Problem	User Action
Unable to connect to Oracle.	■ Ensure Oracle is running. ■ Ensure the ORACLE_SID environment variable is correctly set.
A DBMS upgrade has just been performed.	■ Ensure the IT scripts have correct DBMS link libraries after upgrade. ■ Ensure the DBMS directory references in IT scripts are correct after upgrade. ■ Ensure the precompilers have not changed. ■ Ensure the $IEFH/mkdbs 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.
The following unresolved symbol is received: `osntlispx`	Add $(ORACLE_HOME)/spx/libspx.a to the link.

Application Execution Troubleshooting Without Using the Application Execution Facility

Problem	User Action
Nothing happens after entering a transaction code.	■ Check $AEPATH. ■ Check for aeenv file with trancode in $AEPATH.
When an application screen draws, it shifts or scrolls 1 or 2 lines.	■ The terminal setup file is wrong. Add or modify the p3270keys to include the correct lines= entry for your terminal type. ■ The terminal has a setup capability. Press the key sequence for setup and look for a scroll capability. If there is one that says jump, change it to smooth.

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 correctly 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 AEF. ■ Ensure that the Diagram Trace Utility started. ■ Ensure that 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

Some general technical tips are:

1. It is important to closely adhere to guidelines for software version requirements, for example DBMS and C versions.
2. Separate AEPATH paths and directories with a semicolon (;), not a colon(:).
3. Use the following procedure to find the library where an unresolved reference resides.

   a. Enter the Bourne shell, if necessary.

      ```
      sh
      ```

   b. Compose the following script:

      ```
      for i in 'find / -name '*.a''
      do
      ar -t $i "missing_name".0
      echo $i
      done
      ```

      This script searches all libraries for the missing name. The echo statement lists each name. When the missing name is found in a library, it is listed just before the library name. Otherwise, the system returns the message:

      ```
      Not found.
      ```

When generating a cooperative application with a UNIX or Linux server, generate the client and server separately because the client is local and the server is remote.

Chapter 12: Application Versioning in UNIX and Linux IT

The Implementation Toolset includes a set of files to link version information into applications generated with CA Gen, particularly load module executables, cascade libraries, and operations libraries.

Version information is visible using the supplied $IEFH/make/version_info.ksh procedure on the image or library generated.

The following files are delivered to support application versioning:

- applno.ksh— A shell script in $IEFH/make that is executed during the building of a CA Gen generated application, and is responsible for incrementing the application build count. The results of the build count are stored in an application specific header file that is linked in to the application's load module(s), cascade shared library(s), and operations library(s).
- application.h—A customizable header file initially in $IEFH/include and copied to the application build directory. It defines application-specific information, including product name and product version.
- version.awk—A script in $IEFH/make that dynamically generates a source module that is linked into an operations library to reflect the application version information of other operations libraries linked in to the current operations library being built.
- Version_info.ksh—A shell script in $IEFH/make that will extract versioning information for all CA Gen libraries that are linked into an application. Use of this script will be explained in more detail below.

How Application Versioning Works

A pair of header files are used to represent versioning information that is compiled and linked into CA Gen generated applications. These files are application.h and *_buildver.h.

Application.h contains a set of #defines that refer to information displayed when using the $IEFH/make/version_info.ksh script on the built application files. The customer can edit the #defines in this file to customize an application's version information. The following #define values are customizable in the application.h file:

```
PRODUCT_VERSION - Provides a unique product version number
PRODUCT_NAME - Provides a unique product name
```

*_buildver.h is a generated header file that contains only 1 #define:

```
#define BUILDVERS "xxxx"
```

The BUILDVERS #define refers to the currently incremented build version of the customer's application. This build version is incremented each time the application is built. There is one version of this header file for each load module, cascade and operations library built for a given CA Gen generated application. If needed, the BUILDVERS #define can be changed by the customer to provide an alternate starting build number for the application.

During the application build, the build script checks for application.h in the application's build directory. If they do not exist, it is copied from $IEFH/include. Otherwise, it uses the existing copy that may or may not have been customized in the application’s build directory. Also, if the *_buildver.h file is not present for the load module, cascade, or operations library being built, a new version of the file is built with an initial BUILDVERS value of “00001”.

The shell script applno.ksh, executed during an application build, increments the application's build number in the application's *_buildver.h file. The information in application.h and *_buildver.h is compiled into the generated version stub files gversion.c (for load modules), oversion.c (for operations libraries), or tversion.c (for RI Triggers). The resultant object is then linked into the application or library being built.

The object file created contains a unique string which is identified by executing the $IEFH/make/version_info.ksh script on the executable or library created. When it creates an operations library that uses other operations libraries, the awk script, version.awk, executes to collect the set of unique strings from other operations libraries to be linked into the built operations library. Executing the $IEFH/make/version_info.ksh script on the built operations library will then list other operations libraries that may have been linked into the just built operations library.

Repeated application builds result in an updated build number reflected when executing the $IEFH/make/version_info.ksh script on the rebuilt executable or library.

Using version_info.ksh

The shell script version_info.ksh is delivered in $IEFH/make and is used to display versioning information for all CA Gen libraries that are linked into a CA Gen generated application. The version_info.ksh script is invoked as follows:

```
$IEFH/make/version_info.ksh  <model directory> <load module>
Where:  <model_directory>       is the fully qualified directory where the
application was built
    <load module>               is the name of the load module that you are obtaining
    version information for
```

This script can be used to obtain versioning information on any application or library that was built with the Build Tool, including load modules, cascade libraries and operations libraries.

Applications and libraries evaluated can be built statically (archived) or dynamically (shared).

The following version information is displayed for each CA Gen library listed:

Field name	Description	Example
Title	Short description of the library	CA Gen Translation Shared Library
Version	Delivered version number	VERSION:8.5.0.0
Filename	Actual library filename	FILENAME:libgxlate.sl
Build Number	Library build number. This is incremented if this library is rebuilt	BUILD:85085. For a delivered library, this build number reflects the CA Gen release number (8.5) and the release build number (085th build).
Build Date	Date of library build	DATE:Apr 18 2010 22:00:44

Examples and Explanations

The following examples are executed on an HP Itanium system.

Example #1: Listing CA Gen shared libraries linked to application P301 (with Build Tool token OPT.LIBTYPE=SHARED). In this example, the application.h file was modified to set PRODUCT_VERSION to 2.1.0.0 prior to the application build. This version number will be reflected in the application's load module and cascade library version information.

```
$IEFH/make/version_info.ksh /home/pttest/coop_model P301
CA Gen Library Version List Script
The following CA SHARED libraries are linked into the Load Module file
    P301
Residing in the inqload subdirectory of
    /home/pttest/coop_model
Version and build numbers are included in these listings.
LOAD MODULE:

CA Gen Customer Application Load Module VERSION:2.1.0.0 FILENAME:P301 BUILD:00018
DATE:Aug 19 2010 16:15:21

SHARED TRIGGERS AND/OR ARCHIVE CBD LIBRARIES:

CA Gen Customer Application Trigger Library VERSION:2.1.0.0 FILENAME:libRICOOP
BUILD:00004 DATE:Jun 23 2010 15:39:30

SHARED GEN LIBRARIES:

CA Gen Translation Shared Library VERSION:8.5.0.0 FILENAME:libgxlate.sl BUILD:85085
DATE:Apr 18 2010 22:00:44

CA Gen Multibyte Shared Library VERSION:8.5.0.0 FILENAME:libmbyte.sl BUILD:85085
DATE:Apr 18 2010 21:55:52

CA Gen User Exits Shared Library (C) VERSION:8.5.0.0 FILENAME:libae_userexits_c.sl
BUILD:85085 TARGET:exits DATE:Apr 18 2010 22:13:19

CA Gen Runtime Shared Library (C) VERSION:8.5.0.0 FILENAME:libae_common_c.sl
BUILD:85085 TARGET:c DATE:Apr 18 2010 22:13:18

CA Gen Oracle Functions Shared Library VERSION:8.5.0.0 FILENAME:libae_oracle.sl
BUILD:85085 TARGET:db DATE:Apr 18 2010 22:13:20

CA Gen Decimal Precision Shared Library VERSION:8.5.0.0 FILENAME:libdprt.85.
BUILD:85085 DATE:Apr 18 2010 22:10:59

CA Gen Transactional Execution Env Shared Library (C) VERSION:8.5.0.0
FILENAME:libae_tx_c.sl BUILD:85085 TARGET:middle DATE:Apr 18 2010 22:13:22
```

Example #2: Listing CA Gen archived libraries linked to application P301 (with Build Tool token OPT.LIBTYPE=ARCHIVE). The application.h file was modified with PRODUCT_VERSION set to 2.1.0.0.

```
[176] % $IEFH/make/version_info.ksh /home/pttest/coop_model P301
CA Gen Library Version List Script
The following CA ARCHIVE libraries are linked into the Load Module file
    P301
Residing in the inqload subdirectory of
    /home/pttest/coop_model
Version and build numbers are included in these listings.

LOAD MODULE:

CA Gen Customer Application Load Module VERSION:2.1.0.0 FILENAME:P301 BUILD:00019
DATE:Aug 19 2010 16:12:58

ARCHIVE TRIGGERS AND/OR ARCHIVE CBD LIBRARIES:

CA Gen Customer Application Trigger Library VERSION:2.1.0.0 FILENAME:libRICOOP
BUILD:00005 DATE:Aug 19 2010 16:34:07

ARCHIVE GEN LIBRARIES:

CA Gen Runtime Archive (C) VERSION:8.5.0.0 FILENAME:libae_common_c.a BUILD:85085
TARGET:clib DATE:Apr 18 2010 22:13:16

CA Gen Transactional Execution Env Archive (C) VERSION:8.5.0.0 FILENAME:libae_tx_c.a
BUILD:85085 TARGET:/pldv1/bldwrk/gen/80/base/runtime/rt/build DATE:Apr 18 2010
22:12:37

CA Gen Decimal Precision Archive VERSION:8.5.0.0 FILENAME:libdprt.85.a BUILD:85085
DATE:Apr 18 2010 22:10:59

CA Gen Oracle Functions Archive VERSION:8.5.0.0 FILENAME:libae_oracle.a BUILD:85085
TARGET:/pldv1/bldwrk/gen/80/base/runtime/rt/build DATE:Apr 18 2010 22:12:35

CA Gen User Exits Archive (C) VERSION:8.5.0.0 FILENAME:libae_userexits_c.a
BUILD:85085 TARGET:cxlib DATE:Apr 18 2010 22:13:18

CA Gen Multibyte Archive VERSION:8.5.0.0 FILENAME:libmbyte.a BUILD:85085 DATE:Apr 18
2010 21:55:52

CA Gen Translation Archive VERSION:8.5.0.0 FILENAME:libgxlate.a BUILD:85085 DATE:Apr
18 2010 22:00:44

SHARED GEN LIBRARIES:
```

Chapter 13: The Application.ini File

Overview

The application.ini file stores application-specific environment variables in a 'token=value' pairing. CA Gen delivers this initialization file with the C Runtime in the Gen install directory ($IEFH). 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 cooperative servers and block mode applications, the C Runtime library opens and reads the application.ini file in the model's source directory and sets and uses 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 communication with the Diagram Trace Utility.	1
TRACE_HOST	Defines the Windows system on which the Diagram Trace Utility runs.	<none>
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'.

Chapter 14: Rebuildable Library and Executable Build Counts

As part of the IT installation, there are several delivered libraries and executables that are rebuildable by the customer. These libraries and executables include User Exits, DBMS libraries, and DBMS Loader executables. To help identify whether or not one of these files have been rebuilt, CA Gen will now increment a build counter that will be displayed when executing the version_info.ksh script supplied in the $IEFH/make directory. This feature is similar to that of Application Versioning.

When delivered, the libraries and executables that are rebuildable already have a build number that reflects the release it is part of. For example, when executing $IEFH/make/version_info.ksh on an application load module that was built with the delivered archive version of the user exits library libae_userexits.a, the "BUILD" field will have a build number similar to "85085", which indicates that this library came from the CA Gen 8.5 release, and it was part of the 085th build.

When a rebuildable library or executable is rebuilt, the build number is incremented in such a way to preserve the delivered build number and also reflect the incremented rebuild number. If we refer to the previous example, if the archive version of the user exits library libae_userexits.a was rebuilt twice, the "BUILD" field will reflect the rebuild with "85085-002".

You will also notice that the DATE field has been updated to reflect the rebuild date.

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

The list of IT libraries and executables that will support the build count incrementer upon a customer rebuild include:

Library or Executable	Description	Rebuild Procedure
libae_db2.{ext}	Server DB2 library	mkdbs
libae_oracle.{ext}	Server Oracle library	mkdbs
aefsecex	Security Exit	mksecex
libae_userexits.{ext}	Server User Exits	mkexits
tidb2ddl.exe	Db2 DDL loader Executable	mkdbs
tioraddl.exe	Oracle DDL loader Executable	mkdbs
libcsuvn.**xx**.{ext}	CSU Version User Exit	csuglvn.{plat}
libmqscx.**xx**.{ext}	MQ Series Client User Exit	cmqsexit.{plat}

Library or Executable	Description	Rebuild Procedure
libmqssx.**xx**.{ext}	MQ Series Server User Exit	smqsexit.{plat}
libprex.**xx**.{ext}	C Proxy User Exit	proxyxit.{plat}
libtcpcx.**xx**.{ext}	TCP User Exit	ctcpexit.{plat}
libtxcx.**xx**.{ext}	Tuxedo Client User Exit	ctuxexit.{plat}
libtxsx.**xx**.{ext}	Tuxedo Server User Exit	stuxexit.{plat}
libtxwcx.**xx**.{ext}	Tuxedo WS Client User Exit	ctuxexit.{plat}
Libwscx.xx.{ext}	Web Services User Exit	cwsexit.{plat}

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 Exits Reference Guide*.

More information:

Application Versioning in UNIX and Linux IT (see page 85)

Index

R

S

T

U