CA Gen

NonStop Implementation Toolset User Guide

Release 8.5

Second Edition

CA Technologies Product References

This document references the following CA Technologies products:

- CA Gen

Contact CA Technologies

Contact CA Support

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

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

Providing Feedback About Product Documentation

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

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

Documentation Changes

The following documentation updates have been made since the last release of this documentation.

The following new topics/chapters are added to include the SQL/MP and SQL/MX database information:

- Build Tool (see page 11)
- Using the Build Tool in CA Gen (see page 55)

Updated topics/chapter to include the SQL/MP and SQL/MX database information:

- NonStop Implementation Toolset (see page 9)
- Setup Tool (see page 11)
- CA Gen Runtime (see page 11)
- Compile, Link, and Store an External Action Block (see page 37)
- Using the NonStop Setup Tool in CA Gen (see page 39)
- Using the Applini File (see page 129)

Contents

Chapter 1: Introduction

NonStop Implementation Toolset

The Implementation Toolset (IT) is a collection of tools that lets you build and run NonStop SQL/MP and SQL/MX applications that CA generates. SQL/MP applications are run under the Guardian operating system environment and SQL/MX applications are run under the OSS (Open System Services) operating system environment. Both types of applications are executed using the Pathway TP monitor.

The CA Gen tools do not call out the different operating systems when setting the target environments but instead differentiate the environments by what database (SQL/MP or SQL/MX) is chosen.

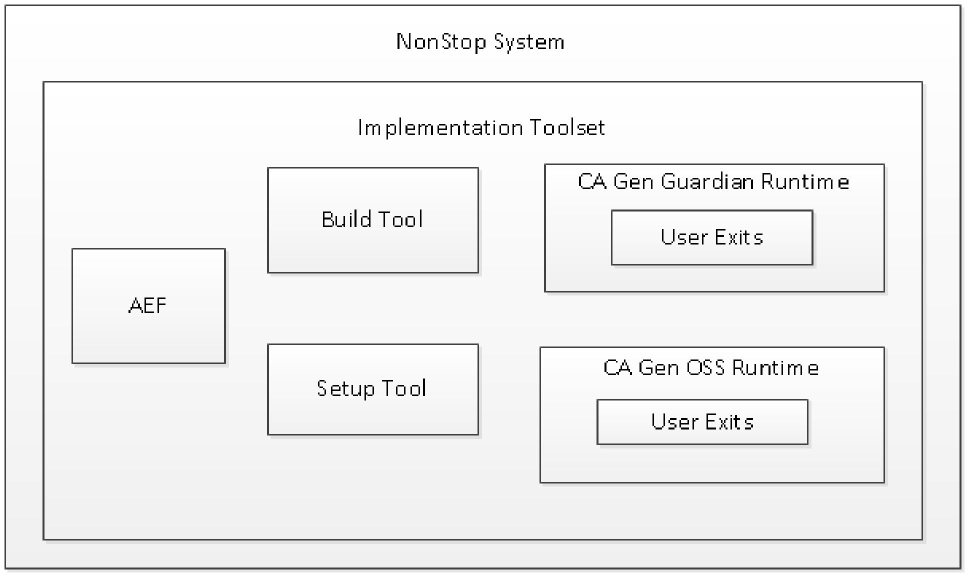
Notes: In this document:

- The term *SQL/MP* refers to the operating system environment is Guardian.
- The term *SQL/MX* refers to the operating system environment is OSS

The tools that comprise the IT are:

- Application Execution Facility (AEF)
- Setup Tool (for building SQL/MP applications)
- Build Tool (for building SQL/MX applications)
- CA Gen Runtime for SQL/MP applications
- CA Gen Runtime for SQL/MX applications
- CA Gen User Exits

Each component of the IT is installed on the target system. CA Gen interfaces with the NonStop operating system for information like name of the terminal, names of Pathway environment components, and the location of CA Gen components. The following illustration shows this relationship:



Note: Users must have the knowledge to configure the development platform to implement generated applications. Users must be familiar with Pathway, SQL/MP, SQL/MX, RSC/MP, and other technologies available on NonStop.

Application Execution Facility

The Application Execution Facility (AEF) lets CA Gen applications execute on NonStop. It functions as an interface between the operating system and the application being run. Since AEF is designed specifically for the NonStop environment, you can take full advantage of the following fundamental features:

- Loosely coupled architecture
- Message-based distributed processing
- Multi-processor parallel execution
- Multi-nodal network-based application deployment
- Linear expandability
- Fault-tolerant application execution
- TMF-based referential integrity

More information:

Testing and Running Applications in NonStop (see page 61)

Setup Tool

The Setup Tool is a Pathway-based application that allows you to deploy applications generated by a CA Gen development environment targeted for NonStop SQL/MP. To compile the source code for deployment to NonStop SQL/MX, use the Build Tool.

More information:

Using the NonStop Setup Tool in CA Gen (see page 39)

Build Tool

The Build Tool is a Java-based application that allows you to build applications generated by a CA Gen development environment targeted for Nonstop OSS.

CA Gen Runtime

The CA Gen Runtime is a set of libraries called by the generated application to provide all the supporting functionality needed by the generated code. Support includes, but is not limited to, date and time functions, string manipulation, dialog manager functions, screen I/O, screen formatting, field validation, 3270 data stream support, and so on.

Separate libraries are included to support block mode and distributed processing. Additionally, the runtime provides an interface between the generated code and the AEF. There is a set of libraries delivered to run on NonStop SQL/MP applications, and another set delivered to run on NonStop SQL/MX applications.

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.

More information:

Implementing CA Gen User Exits in NonStop (see page 99)

Audience

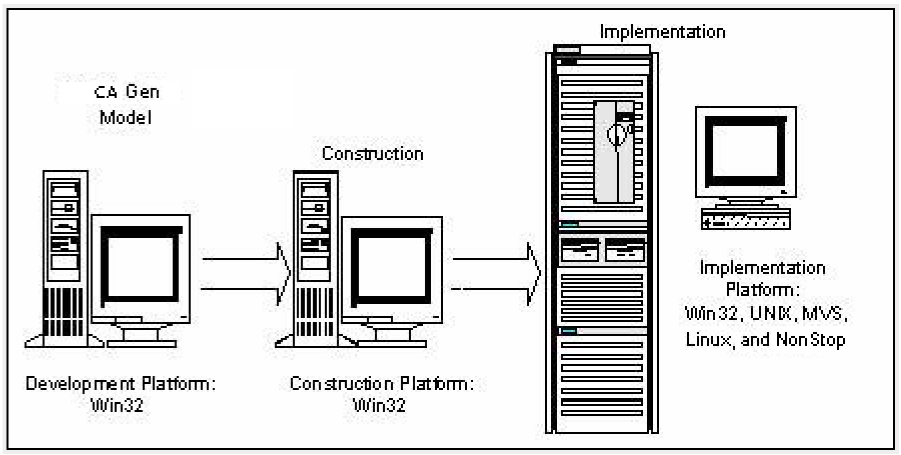
This guide is intended for CA Gen developers who use the Implementation Toolset on NonStop to build and install CA Gen generated applications.

Users should have the knowledge to configure the development platform to implement generated applications. Users must be familiar with Pathway, SQL/MP, SQL/MX, RSC/MP, and other technologies available on NonStop.

Chapter 2: Prepare a CA Gen Model for Multiple Platforms on NonStop

Target System Implementation

Target system implementation is a 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:


CA Gen Model
Construction
Implementation
Development Platform: Win32
Construction Platform: Win32
Implementation Platform: Win32, UNIX, MVS, Linux, and NonStop

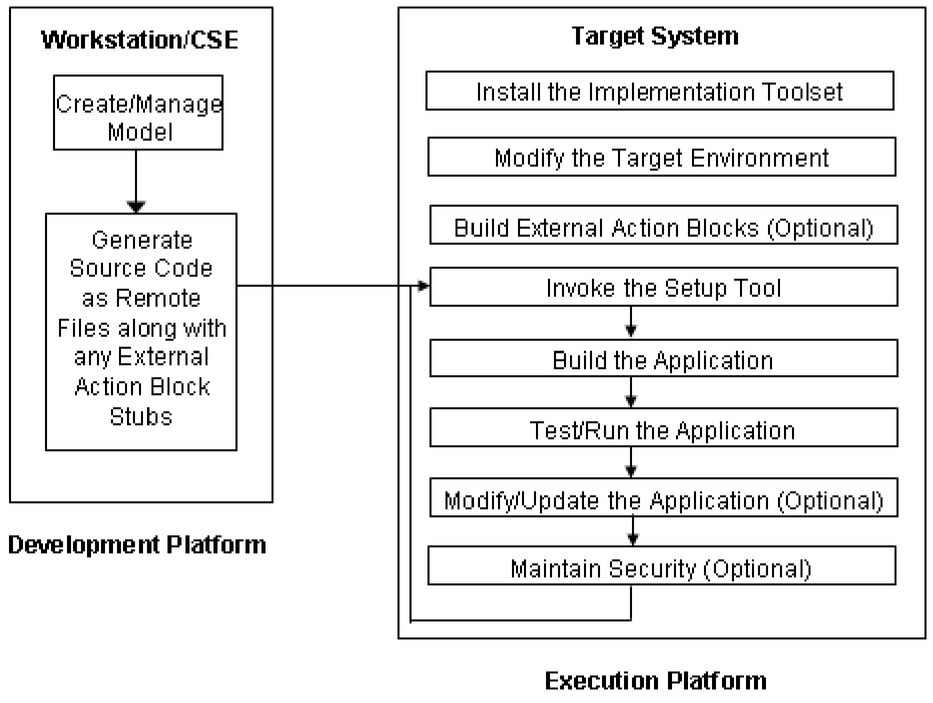
Applications are developed as models using various CA Gen Toolsets on a workstation called the Development Platform. Models are prepared for execution through a process called *construction*, which may occur on the development platform or on another system. The resulting components, called remote files, 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 then test and run the application on the target system within the CA Gen environment. In the case of NonStop, each application is comprised of at least three remote files—one for the database information, one for the referential integrity triggers, and 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 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*.


Workstation/CSE
Create/Manage Model
Generate Source Code as Remote Files along with any External Action Block Stubs
Development Platform
Target System
Install the Implementation Toolset
Modify the Target Environment
Build External Action Blocks (Optional)
Invoke the Setup Tool
Build the Application
Test/Run the Application
Modify/Update the Application (Optional)
Maintain Security (Optional)
Execution Platform

Note: This illustration is referenced several times throughout the guide to illustrate the relationship between tasks required to use the IT on the target system.

More information:

Remote Files (see page 15)

Packaging

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, each load module is implemented as a separate remote file and linked to the target system into a separate executable. Component Models are packaged into one or more operations libraries.

Note: For more information about packaging, see the *Workstation Construction User Guide*.

Remote Files

After a completed model goes through construction, it is stored as a compilation of source code, data definition language (DDL), and other components in special files called *remote files*, also known as implementation packages (IPs). The file extension of a remote file is .rmt. Sections in the remote files define the organization and contents of the CA Gen generated application, which makes it possible for CA Gen to identify and process all the relevant components of the application on the platform where the application will execute.

After construction is complete, the remote files are moved (outside of CA Gen) to the target system where they can be stored, interpreted, compiled, tested, and executed.

Note: Do not edit the remote file with a PC text editor before transferring them to NonStop. Some editors embed extra characters within the text. NonStop's compilers may reject the generated code.

Types of Remote Files

The following types of remote files are required for an application:

- Load Module
- Database
- Referential Integrity

Note: Operation Libraries are not supported on NonStop.

These remote file types are summarized in the following table and explained in more detail in the following sections:

Type	Number Used	Description of Contents
Load Module	One or more per application	Application source code for a single load module.
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, which 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 data source.

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 data source. This lets each procedure step reference data in its input view without considering how the data values are obtained.

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

Database Remote Files

The database remote file contains DDL statements and installation information. You can 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.

Transferring Remote Files to the Target System

Remote files, which may or may not contain external action block stubs, are moved from the development platform to the target system using the file-transfer process you choose.

Ensure care when transferring remote files from the Workstation to the NonStop server. Even though the files are ASCII text files, transfer them in BINARY format. This is due to a restricted length in the NonStop Edit format file structures. CA Gen occasionally generates text lines longer than the 255 character capacity of the NonStop Edit format file structure.

Note: If files are transferred in ASCII format, anything more than 255 characters is truncated without warning. This truncation will cause problems later during compilation. Some file transfer programs, including FTP in text-mode, truncate large ASCII-based files.

Regardless of the number of files in the development platform's subdirectories, transfer files only with a file extension of .rmt. If more than one load module remote file has been generated, each one must be transferred to the target platform.

The load module, DDL, and RI trigger remote files can be transferred directly to the appropriate installation directories on the target system, or they can be transferred to a subvolume and later moved to the installation subvolume.

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.

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.

More information:

Create and Test External Action Blocks on NonStop (see page 31)
How EABs are Created (see page 32)

CA Gen Runtime User Exits

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

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

Chapter 3: Prepare the Target Environment

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

- Pregeneration tasks
  - Packaging EABs
  - Considerations for Remote Generation
  - Considerations for Trace Generation
  - Considerations for Performance
- Prebuild 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 pregeneration 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 in 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 a 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

Because CA Gen manipulates data according to the type of targeted DBMS, existing data may 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 is complete. 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.

Referential Integrity 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 may 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 on 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.

Prerequisite Tasks on the Target System

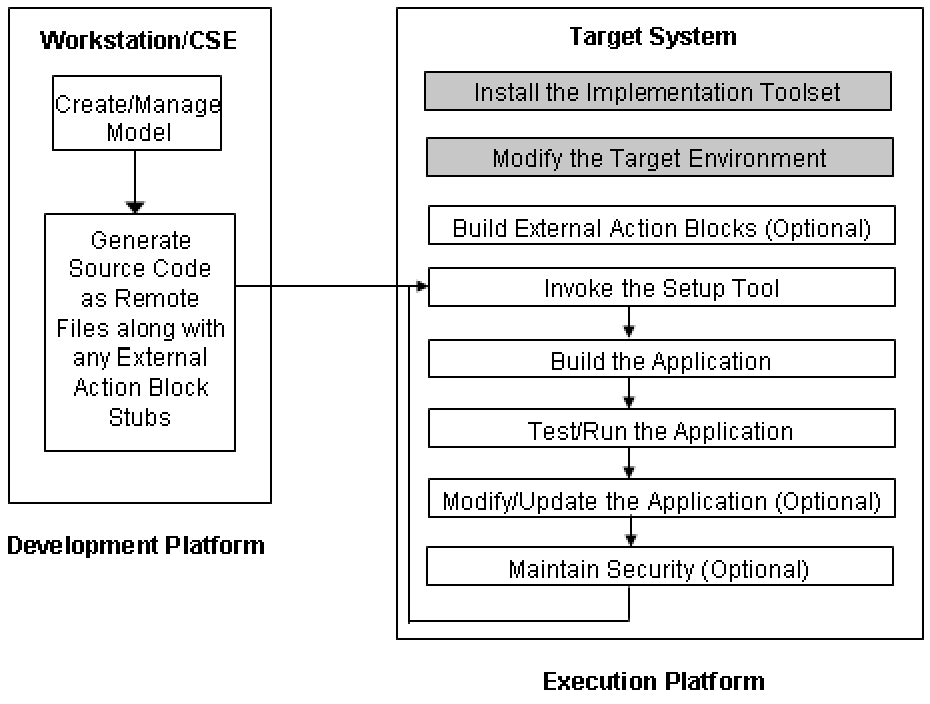
You must perform the following tasks to successfully implement a generated application on your target system:

- 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:


Workstation/CSE
Create/Manage Model
Generate Source Code as Remote Files along with any External Action Block Stubs
Development Platform
Target System
Install the Implementation Toolset
Modify the Target Environment
Build External Action Blocks (Optional)
Invoke the Setup Tool
Build the Application
Test/Run the Application
Modify/Update the Application (Optional)
Maintain Security (Optional)
Execution Platform

Install 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 IT, Setup Tool, and Build Tool

To facilitate application security, the CA Gen administrator super user must own all IT runtime subvolumes, Setup Tool subvolumes, and files. The IT runtime and Build Tool directories containing the deliverables for NonStop SQL/MX must also be owned by the super user.

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

User Modifiable Files

A number of user modifiable files are delivered with the IT. These files include initialization files, sample files, and user exit files.

Important! You are responsible for retaining changes to the user modifiable files in case you reinstall the existing installation so that changes are not lost.

The following modifiable files are delivered with the CA Gen IT that supports NonStop SQL/MP:

- applini
- codepage
- ntcpr1s
- ntcpr2s
- t3270es
- t3270ns
- t6530es
- t6530ns
- termcnfg
- dpsuetdm
- tirdcryp
- tirdlct
- tirdrtl
- tirelog
- tirhelp
- tirmtqb
- tirncryp

- tirsecr
- tirsecv
- tirsysid
- tirterma
- tirupdb
- tiruppr
- tirurtl
- tirusrid
- tirxlat
- tiryyx

The following modifiable files are delivered with the CA Gen IT that supports NonStop SQL/MX:

- application.ini
- codepage.ini
- tirdcryp.c
- tirdlct.c
- tirdrtl.c
- tirelog.c
- tirhelp.c
- tirmtqb.c
- tirncryp.c
- tirsecr.c
- tirsecv.c
- tirserrx.c
- tirsysid.c
- tirterma.c
- tirupdb.c
- tiruppr.c
- tirurtl.c
- tirusrid.c

- tirxlat.c
- tiryyx.c

Define OSS Environment Variables

The IT components that were installed to support NonStop SQL/MX use environment variables to specify paths and other configuration information necessary to build 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 that 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 displayed in the following example:

```
echo $IEFH
```

Required OSS Variables

The following table describes the set of environment variables required to use the IT for NonStop SQL/MX:

Name	Description
IEFH	CA Gen IT home directory for NonStop SQL/MX. This is the directory used to locate the runtime libraries and the Build Tool. **Example:** export IEFH=/opt/gen85
IEFPLATFORM	Name of hardware platform. This needs to be set to IEF_NONSTOP.
IEFGXTP	Specifies the directory that contains the codepage translation files. **Example:** export IEFGXTP=$IEFH/translat
SQLMX_HOME	Location of the SQL MX Preprocessor. **Example:** export SQLMX_HOME=/usr/tandem/sqlmx

Name	Description
PATH	Search path for executables. This must include $IEFH/bin, $IEFH/bt, and SQLMX_HOME. **Example:** export PATH=$IEFH/bin":"$IEFH/bt":"$SQLMX_HOME/bin":"$PATH

Chapter 4: How to Create and Test External Action Blocks on NonStop

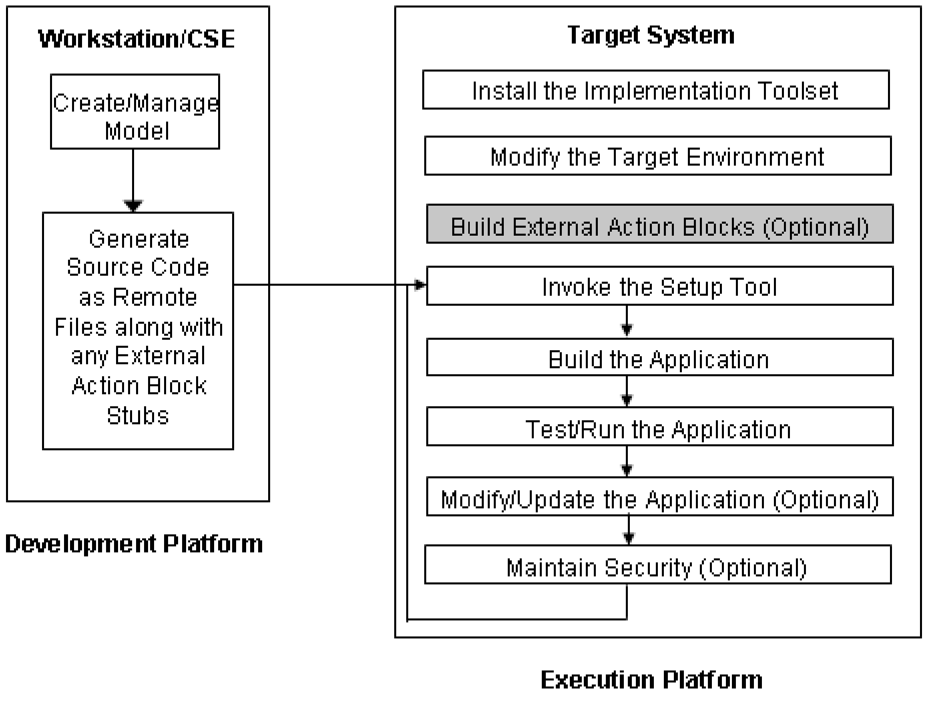
Create and Test External Action Blocks on NonStop

An External Action Block (EAB) defines the interface between a procedure step or action block that invokes it and the logic created outside of CA Gen (that is, handwritten subroutines). It provides the structure that is used to match the views of a procedure step with the views (input and output arguments) of the handwritten subroutine. This may be necessary when establishing communications between the following entities:

- A non-TCP requester and a CA Gen generated application
- A CA Gen generated application and application code not generated by CA Gen
- CA Gen and non-SQL data sources (such as tape drives, communications lines, enscribe files, and so on)

EABs are used on the target system and not on the development platform; EAB stubs are defined on the development platform.

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


Workstation/CSE
Create/Manage Model
Generate Source Code as Remote Files along with any External Action Block Stubs
Development Platform
Target System
Install the Implementation Toolset
Modify the Target Environment
Build External Action Blocks (Optional)
Invoke the Setup Tool
Build the Application
Test/Run the Application
Modify/Update the Application (Optional)
Maintain Security (Optional)
Execution Platform

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

EABs are created when code is generated. The CA Gen software cannot generate actual code for EABs, but 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 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, copy the EAB file to the target system where the appropriate logic must be added to the generated action block stub before it is compiled and installed. You must perform the following steps before you build and implement the load module containing the EAB:

- Identify EABs
- Locate EABs
- Analyze EABs
- Create external action logic
- Compile, link, and store EABs

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 as well as 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, complete 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 the EAB library.

Identify 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=EXTERN1
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 in the same way 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).

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.

Handle 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 in 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 must adhere to the following rules:

- The character representation of the number placed in the DPrec array may contain fewer digits than are defined for the attribute. However, it must not contain more digits to the left or right of the decimal than are 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)

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, 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.

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

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 the PSMGR-EAB-DATA array, 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 new subroutines called by the stub).
- Link the EAB into an object library.
- Using Setup tool, define the fully qualified file name for the object library through the Location Details screen of the Setup Tool.

  Note: For more information about the Location Details screen, see the *Setup Tool Help*.
- Using Build Tool, define the fully qualified file name for the object library through the LOC.EXTERNAL_LIB token used by the Build Tool.

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

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.

Test an External Action Block

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

More information:

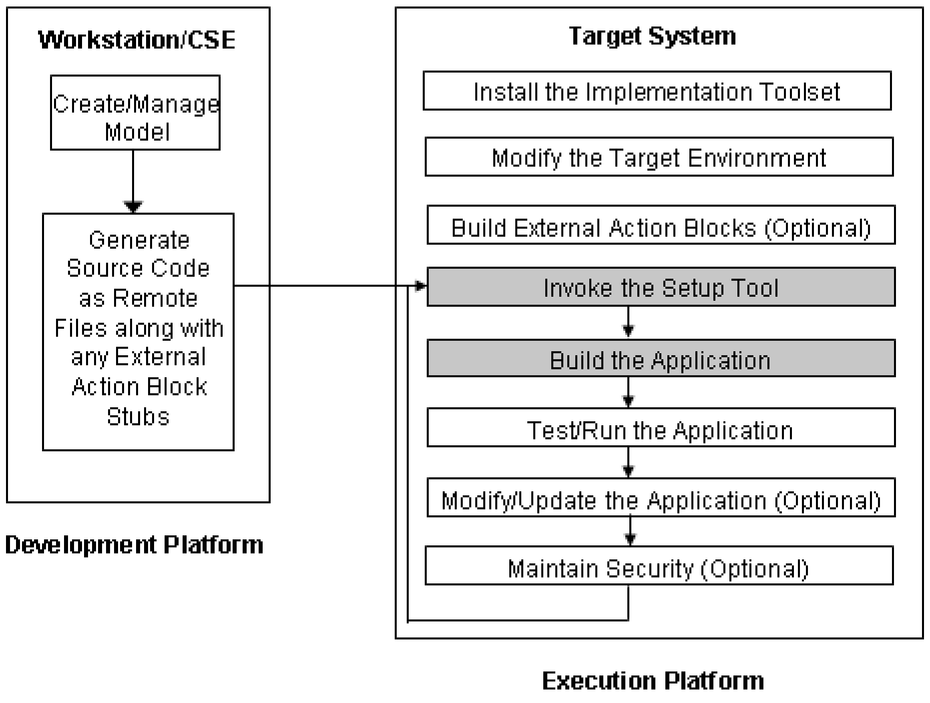
Testing and Running Applications in NonStop (see page 61)

Chapter 5: Using the NonStop Setup Tool in CA Gen

Set up the target environment and build user exits before processing remote files. Compile external actions blocks and place them in the appropriate location. You must use the Setup Tool to create a target configuration and build CA Gen generated applications for NonStop SQL/MP. When the application is being built, the remote files are validated by checking their control information; if valid, this information is used along with the target configuration information to complete building the application.

Note: For more information about building applications targeting NonStop SQL/MX, see Using the Build Tool in CA Gen (see page 55).

The relationship of processing remote files to the other required and optional IT tasks are described in the following illustration:


Workstation/CSE
Create/Manage Model
Generate Source Code as Remote Files along with any External Action Block Stubs
Development Platform
Target System
Install the Implementation Toolset
Modify the Target Environment
Build External Action Blocks (Optional)
Invoke the Setup Tool
Build the Application
Test/Run the Application
Modify/Update the Application (Optional)
Maintain Security (Optional)
Execution Platform

Setup Tool Terminology

An understanding of the following terms is important to use the Implementation Toolset and Setup Tool successfully:

Target Configuration

Identifies a configuration environment created for a model to run on NonStop. A target configuration is an object in the Setup Tool to describe the environment.

Build

Identifies the process of compiling and linking source code generated by CA Gen.

Split

Unbundles a remote file and make its ICM available for processing.

Implementation Package (IP)

Identifies a remote file. On NonStop, a target configuration is comprised of at least three IPs—a database one (DDL), a referential integrity one (CASCADE), and one or more load modules.

Member

Identifies a single source file within an IP.

The Setup Tool

This section introduces you to the Setup Tool and should be used as a supplement to the *Setup Tool Help*, which is available online.

The Setup Tool compiles CA Gen generated source code that is targeted for NonStop. It is a Pathway application that interacts with you to provide the following functionality:

- Target Management

  You can create and manage target configurations for each model that is being built. Target configuration includes subvolume locations, Terminal Control Process (TCP) and Pathway parameters, and compiler settings.

- Build Control

  You can determine the portions of an application to build or rebuild at any point in the build cycle (Split, Compile, Link).

- Online Status Monitoring

  You can view the status of the build progress. Monitoring occurs at the following levels:

  - Target
  - Implementation Package
  - Member

  You can see the status at each level in terms of a stage and a percentage of that stage on how far the build is complete.

- Standard User Interface

  The Setup Tool conforms to standard NonStop Pathway applications based on the Requester Server model.

- Built-in Security

  The Setup Tool provides an initial logon screen and the ability to secure targets using NonStop's conventional system for securing system objects. For example, you can secure a target for access by members of a login group, by the owner of the target, or by any user in the system.

- External Libraries

  You can link up to ten user-defined external libraries into any target.

- Help Support

  You can access online help from any screen in the Setup Tool. The online help explains every field on a particular screen.

Note: Because the Setup Tool is a Pathway-based application, any file (including SQL objects) created as a result of a CA Gen generated application installation will be owned by the user whose ID is running the Setup Tool's Pathmon process.

Application Member Locations on the Target System

When the Setup Tool builds an application, it splits the transferred remote files into several members and compiles these members into object code. The Setup Tool links the object code into executables. These members, object code, executables, and other support files are placed into a directory-like structure.

On NonStop, the default naming convention of this directory structure contains a $vol.subvol location name as a starting point, also known as a target or base location. The Setup Tool appends a suffix character to the end of the subvol portion of the name to form valid $vol.subvol names for all file locations. Therefore, the target location ($vol and subvol) cannot be more than 15 characters in length including the $ symbol for the volume name.

The following table explains the naming convention and the contents of each subvolume used by the Setup Tool. In the table, $data.app is used as the target or base location.

Suffix	Contents	Description	Example
a	Work area	A temporary work area used during the installation of a target. Files may be created and deleted by the Setup Tool.	$data1.appa
b	RI Library	Contains the RI library (built by the Setup Tool).	$data1.appb
c	RI Source Code	Contains the RI trigger source (split from the RI remote file).	$data1.appc
d	DDL Source Code	Contains the DDL command file used to create the application database (split from the database remote file).	$data1.appd
g	SQL Compiler Listings	Contains listings produced by the SQL compiler	$data1.appg
i	ICM Source Code	Contains the ICM for all remote files (split from all remote files).	$data1.appi
j	RI Object Code	Contains the RI trigger objects for all source files in location c.	$data1.appj
k	Linker Listings	Contains the Linker listings of all the load modules.	$data1.appk
m	MAKE Files	Contains temporary files used by the Setup Tool.	$data1.appm
o	Load Module Object Code	Contains load module object code for all source files in location s.	$data1.appo
p	Code Listings	Contains listings produced by the Native Mode compiler for all source files in location s.	$data1.appp
r	RI Listings	Contains listings produced by the Native Mode compiler for all source files in location c.	$data1.appr
s	Load Module Source Code	Contains the source files for each member of a load module (split from the load module remote files).	$data1.apps
t	Database SQL Tables	Contains the SQL database for the application (built by the Setup Tool).	$data1.appt

Suffix	Contents	Description	Example
u	Pathway Server Configuration Files	Contains the server configuration files for the CA Gen generated application (built by the Setup Tool).	$data1.appu
x	Load Module Executables Application's copy of applini file	Contains the application executables as specified in the server configuration files. This file contains the Diagram Trace Utility settings.	$data1.appx
NONE	Files used in order to execute the application built.	Contains a set of files that have been copied from the CA Gen IT or created by the Setup Tool so that the application can be executed.	$data1.app

In addition, the target location (that is, $data1.app) contains Pathway configuration files, the codepage file, the AEF startup macro, the droptarg TACL macro, and the application's SQL catalog (if you use the default catalog location).

You can change all these component locations before installing the application according to your requirement.

Start the Setup Tool Pathway

The Setup Tool is a Pathway application. You must start its Pathway environment before using it to install CA Gen generated applications.

Follow these steps:

1. Set your current volume to the location where the Setup Tool is installed.

   ```
   VOLUME $<setvol>.<setsubvol>
   ```

2. Start the Setup Tool Pathway environment.

   Note:

   If you are running the Setup Tool *for the first time*, you need to cold load the Pathway environment by issuing the SETCOLD command.

   If this is *not the first time* the Setup Tool is being loaded, then cool load the Pathway environment by issuing the SETCOOL command.

   The Setup Tool is now ready for use.

Stop the Setup Tool Pathway

If the Setup Tool Pathway environment is no longer needed, you can stop the Setup Tool.

Follow these steps:

1. Set your current volume to the location where the Setup Tool was installed.

   ```
   VOLUME $<setvol>.<setsubvol>
   ```

2. Stop the Setup Tool by issuing the SETSTOP command.

   The Setup Tool Pathway is stopped.

Install a CA Gen Generated Application in NonStop

An application generated by CA Gen and targeted for NonStop must contain a database remote file, an RI triggers remote file, and at least one load module remote file.

The following sections explain the activities that you must follow to build an application for the first time:

1. Log on to the Setup Tool
2. Create a target configuration
3. Build the application
4. Review the build results

Note: It is assumed that the Setup Tool has already been started at this point.

More information

Testing and Running Applications in NonStop (see page 61)

Log On to the Setup Tool

Logging on to the Setup Tool is the first step in building an application.

Follow these steps:

1. Set your current volume to the location where the Setup Tool is installed by issuing the following command.

   ```
   VOLUME $<setvol>.<setsubvol>
   ```

2. Enter the SETUP command.

   The main splash screen appears.

   Note: After you log on to the Setup Tool, you have access to the online help system. From the main splash screen, you can press the SF1 function key to access the main menu online help. If you press the SF1 function key again, the online help's Table of Contents is displayed. The Table of Contents provides information for all the features in the Setup Tool.

3. Enter your TACL logon credentials and press the F1 function key. You may use your UserId or Safeguard alias. Both UserIds and aliases must conform to Safeguard naming conventions.

   You are logged on to the Setup Tool and the main menu appears.

Note: You can press the F1 function key on any screen (except for the main splash screen) to access online help for fields on that particular screen.

Create a Target Configuration

Creating a target configuration is the second step in building an application.

Follow these steps:

1. Press the F1 function key from the main menu.

   The Target Configuration screen appears.

2. Enter the configuration details. Optionally, you can modify the Pathway parameters, compiler options, and location details. Defaults have been provided for these settings.

3. Press F4 to save the target configuration.

   The target is configured with your settings.

Note: The Setup Tool configures certain Pathway parameters based on the number of servers being installed.

More information:

Modify Pathway Parameters (see page 46)
Modify Compiler Options (see page 46)
Modify Location Details (see page 47)
Modify EAB Locations (see page 47)

Modify Pathway Parameters

You can accept the default parameter settings in the Target Configuration screen or update the parameters according to your requirements.

Follow these steps:

1. Press the F5 function key to modify the default Pathway parameters.
2. Press the F6 function key to modify the Pathway TCP parameters.
3. Press the F7 function key to modify the Pathway Server parameters.
4. Press the F4 function key to save changes.
5. Press the F2 function key to return to the Target Configuration screen.

   The target configuration contains the updated Pathway parameters.

Modify Compiler Options

You can accept the default compiler options in the Target Configuration screen or update the options according to your requirements.

Follow these steps:

1. Press the F7 function key to modify the default compiler options. You can modify the C compiler options and the SQL compiler options.
2. Press the F4 function key to save changes.
3. Press the F2 function key to return to the Target Configuration screen.

   The target configuration contains the updated compiler options.

Modify Location Details

You can accept the default target locations in the Target Configuration screen or update the locations according to your requirements.

Follow these steps:

1. Press the F6 function key to modify the default target locations. You can modify the $vol.$subvol locations for the set of files that are split out of the remote files and created as a result of the build.
2. Press the F4 function key to save changes.
3. Press the F2 function key to return to the Target Configuration screen.

   The target configuration contains the updated location details.

Modify EAB Locations

You can add EAB locations according to your requirements.

Follow these steps:

1. Press F6 from the Target Configuration screen.
2. Press the F5 function key on the Location Details screen to provide an EAB library or libraries that will be bound into an application. You can add the $vol.$subvol.file name for the EAB library that you need for the build.
3. Press the F4 function key to save changes.
4. Press the F2 function key to return to the previous menu from where you navigated to the Location Details screen.

   The target configuration contains the updated EAB location details.

Build the Application

You must perform the following steps in the Target Configuration screen to build an application.

Follow these steps:

1. Press the SF8 function key after you update the target configuration details.

   The Install IPs screen appears. A list of IPs for the specified target will be listed.
2. Enter Y in the Select All IPs field if this is the first time the application is being installed. This will build all IPs. For subsequent installations, see Rebuild an Application. If you need to build a subset of the set of IPs, tab to each IP of choice and enter an X to select.

3. Press the F4 function key to start building the application.
4. Press the F5 function key to display the Install Status screen where you can monitor the progress of the build.

   The application is built as displayed in the Install Status screen.

The following events occur during the build:

- The base environment for the target is built. This includes setting up an application SQL catalog if none exists. Files and programs are also copied from the IT to your target location.
- The remote files are validated by verifying that their control information matches the information for the target configuration (language, DBMS, and so on). The Setup Tool database is then populated with that information.
- The remote files are split into their various component parts (procedure steps, actions blocks, and so on) and stored in their appropriate locations.
- The source code is compiled into object code.
- The object code is linked with the appropriate libraries to produce the executable load module.
- The transaction table is loaded with application trancodes.

Review the Build Results

You can view the build results after the application is built. This helps you identify problems when builds fail.

Follow these steps:

1. Press the F3 function key from the main menu.

   The Install Status screen appears.
2. Select the target you need to review.
3. Press the F8 or F9 function key to traverse down any of the IPs to view details of the build results, including compilation listings, binding listings, SQL listings, EMS log files, TMT load listings, and so on.

More information:

Testing and Running Applications in NonStop (see page 61)

Setup Tool and EMS

By default, the Setup Tool uses its own EMS template files to correctly display messages generated during the installation of CA Gen applications within its own EMS viewer. In this mode and within this viewer other NonStop sub systems' messages that interface with EMS will display with unresolved text. This is because the Setup Tool's internal EMS viewer is unaware about other subsystem's EMS templates. Commenting out the SET SERVER DEFINE for =_EMS_TEMPLATES for the SVIEWEMS server will cause SVIEWEMS's EMSDIST processes to use the non resident template file provided by the operating system, that is, $SYSTEM.SYSxx.TEMPLATE.

Note: Using $SYSTEM.SYSxx.TEMPLATE causes the Setup Tool's EMS viewer to display Setup Tool's related events with unresolved text.

Generally, you should use the INSTTMPL macro supplied with the Setup Tool to merge its EMS templates into the system templates so that all the EMS log retrieval utilities have access to all templates for all sub systems' reporting messages to the EMS log files (through the default $0 collector process). After INSTTMPL macro is run successfully, the Setup Tool's EMS viewer will display all events and messages in a resolved manner.

Note:

- You must be logged on as SUPER.SUPER to run the INSTTMPL macro.
- For more information about installing the Setup Tool EMS templates, see the *Distributed Systems Installation Guide*.

More information:

Testing and Running Applications in NonStop (see page 61)

CA Gen Specific Parameters Included in Pathway Server Configuration Files

When an application is built, a set of Pathway server configuration files are created—one per load module. These files are accessed when the application is cold-loaded using the PWCOLD command. These configuration files contain SET SERVER commands that define the Pathway servers. The Pathway servers created through the Setup Tool contain several additional SET SERVER commands that are specific for CA Gen applications.

The following table lists the additional commands:

Param or Define	Name	Description
Param	charset-id	Specifies the character set identification number to be used with double byte character set (DBCS) applications. This parameter is *not* modifiable.
Param	codepage	Specifies which codepage was used at generation. This information is used internally for translation purposes. This parameter is *not* modifiable.
Param	Iefgxtp	Location of the codepage tables used by applications at runtime to resolve translations between clients and servers. **Note:** This parameter is only used for Distributed Processing Servers.
Param	trace-log	Specifies the level of information logged to an internal trace file. This log file is created in the base target subvolume and is named lg*xxxxxx* (where *xxxxxx* is the process ID of the running server). A value between 1 and 31 is provided—the higher the value, the more information will be traced. A trace value of 31 may consume a large amount of disk space, which can significantly slow processing. Therefore, you must use the trace-log field carefully in production systems. This value should be left commented out for production systems, and should only be used to help debug a problem with the server. By default, this value is commented out. **Note:** This parameter is only used for Distributed Processing Servers. Block Mode servers use the Snap Trace field on the AEF screen.
Param	ptopt	Specifies that flat-file profiling is used with the server. This parameter is *not* modifiable.
Param	pthome	Specifies the location of the flat-file profile that is associated with the server. **Default:** $vol.appm If modified, all servers must specify the same location.

Param or Define	Name	Description
Define	=ovflow	Specifies the name of the overflow file used when large views are passed between procedure steps and cannot be transferred in a single input/output (I/O) operation. The load module that contains the procedure step initiating the flow writes the views into this file after they are compressed. The load module containing the receiving procedure step reads the file define by =ovfile, uncompresses the data, and continues with the execution of the flow. **Default:** $vol.appm.overflw1 If modified, all servers must specify the same file name.
Define	=tmt_e	Tells the DPSTMTC GAEF server where to find the transaction mapping file used by distributed processing server applications. This file is a flat file version of the TMT SQL table. This option is only available for distributed processing server applications. **Default:** $vol.app.tmte If modified, all servers must specify the same file name.
Define	=application_ini	Specifies the fully qualified name of the initialization file used with the Diagram Trace Utility (DTU). This file contains settings that must be modified to enable tracing with the Diagram Trace Utility. By default, the entries in this file are commented out. **Default:** $vol.appx.applini If modified, all servers that intend to communicate with the Diagram Trace Utility must specify the same file name.
Define	=tcpip^process^name	Specifies the name of the TCP/IP process running a pair of Listener and Telnet processes on the system. This TCP/IP process is used by the servers to communicate with the remote Diagram Trace Utility. **Default:** $ZTC0 If modified, all servers that intend to communicate with the Diagram Trace Utility must specify the same TCP/IP process. For the name of this process, contact your system administrator.
Define	=codepage_ini	Specifies the fully qualified name of the codepage translation file. This file contains specifications for code pages used in CA products and generated applications. **Default:** $vol.app.codepage

Rebuild Pathway Configuration Files

You can modify the application's Pathway configuration files any time after your application is built; you do not need to rebuild the application using the Setup Tool.

Follow these steps:

1. Update the Pathway configuration information for the specific application in the Pathway Parameters, TCP Parameters, or Server Parameters screens.
2. Press the SF12 function key to rebuild the Pathway configuration files.

   The Pathway configuration files are rebuilt. For these changes to take effect, you must cold-load your application.

   Important! Using this option after you have edited and updated the Pathway configuration files outside of the Setup Tool will cause all those changes to be lost.

Rebuild a NonStop Application

Application generated using CA Gen can change through time. If a change involves only one piece of code or source member, then you do not need to rebuild the entire application. This also applies in cases where temporary fixes are applied to generated code on the NonStop host.

The Setup Tool can rebuild an application based on dependencies like the UNIX make utility. This means if source code has a newer timestamp than its corresponding object (or it does not exist), then a compilation event is triggered. Likewise, if object code has a newer timestamp than the executable they created (or it does not exist), then a link event is triggered. The same applies to a remote file and its source after it is split.

Let us assume that a temporary fix was applied to certain source members on NonStop. To rebuild your application servers, recompile the affected source members by selecting one or more remote files in the Install IP screen and pressing the F4 function key to reinstall your changes. If you are not sure which remote file contains the affected source members, select all the load module remote files. Even though you selected all load modules, the Setup Tool compiles only those source members that are affected by the fix you just applied. Any source member not affected by the fix will show a status of RETAINED, which means that the Setup Tool retained their original status. Use the *Force installed IPs to Rebuild from Step* field on the IP Install screen to override the dependency logic.

RETAINED Status

The RETAINED status assigned to any source member indicates that the source member was not rebuilt. The following reasons explain why source members can display a RETAINED status during installation:

- Common Action Blocks (CABs) are used when an application contains common code used by different load modules. The source member that implements the CAB needs to be compiled once; it displays a status of COMPILE at the end of the COMPILE stage. However, any reference to the CAB from other load modules will show as RETAINED.
- If there was a problem during the generation or packaging of the remote files on the CA Gen development workstation, some of the source members show a RETAINED status. This generally occurs during a first attempt to install an application on NonStop. Review the way you are packaging the load modules and the generation process. Ensure that your application does not contain CABs.

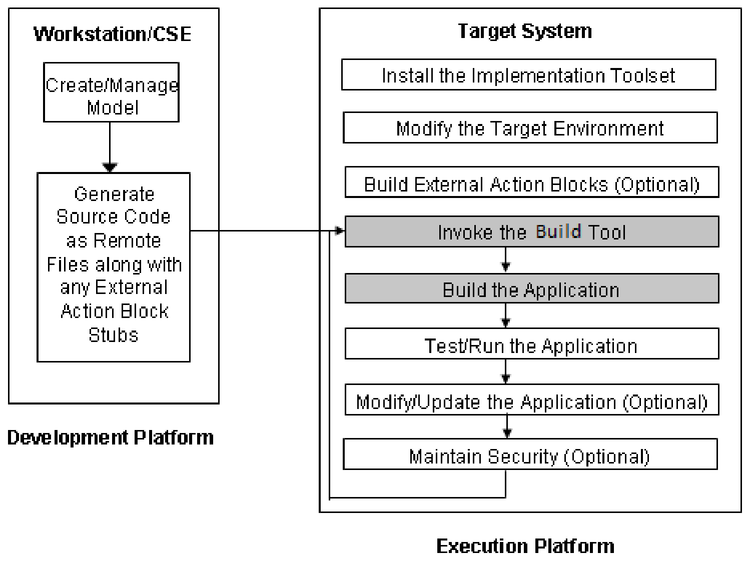
Sentinel Line

When a remote file is split, the Setup Tool appends a comment line at the end of each one of the extracted source members. This comment line is called *Sentinel Line* and it contains information unique about each source member. The Sentinel Line is used as part of the dependency rule during the compilation of the source member. Any modifications to this line may cause unexpected results during any subsequent builds. If you need to manually modify the code, you can do so as long as the Sentinel Line is not modified, and remains the last line of the code.

Chapter 6: Using the Build Tool in CA Gen

To build CA Gen generated applications that target NonStop SQL/MX, use the Build Tool. The Build Tool is a delivered Java application that supports building NonStop SQL/MX applications.

The relationship of processing remote files to the other required and optional IT tasks are described in the following illustration:


Workstation/CSE
Create/Manage Model
Generate Source Code as Remote Files along with any External Action Block Stubs
Development Platform
Target System
Install the Implementation Toolset
Modify the Target Environment
Build External Action Blocks (Optional)
Invoke the Build Tool
Build the Application
Test/Run the Application
Modify/Update the Application (Optional)
Maintain Security (Optional)
Execution Platform

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

A script controls the processing of each load module and RI trigger set. 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 invoking and using the Build Tool, with selecting and customizing scripts, see the *Build Tool User Guide*.

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 more information about release specific instructions, see the *Release Notes.*

Create Target Environment

Once the application is built from the Build Tool, several tasks are performed before the application can be tested, run, or both. A summary of tasks is as follows:

- Create and Register SQL/MX tables and modules
- Create Guardian Target Environment

These tasks are explained in the following sections.

Create and Register SQL/MX Tables and Modules

Several steps are taken to set up the SQL/MX tables to be used with the application only built. The generated remote file representing the database information contains the data definition language use to create the SQL/MX tables. The following steps are taken to create and register the SQL/MX tables using this file:

Create the Catalog and Schema

Use mxci to create the catalog and schema. Use the same values that are given with Build Tool tokens OPT.MODULE_CATALOG and OPT.MODULE_SCHEMA for the catalog and schema names.

Create the Tables

You can obey the INS file found in the database remote file to create the tables. Split the database remote file (using the Build Tool) to extract the INS file.

Register the SQL/MX Modules

Use mxCompileUserModule to register the modules. Use the same values that are given with the Build Tool tokens OPT.MODULE_CATALOG and OPT.MODULE_SCHEMA for the catalog and schema names.

Optionally, building of the application generates two files, MX and MX-INPUT. Using the previous tokens, with OPT.MODULE_LOCATION, these files provide a simple script to perform the previous steps. These scripts can be customized as one sees fit.

Create Guardian Target Environment

Because the application has been built using the Build Tool, several steps are taken to create the Guardian target environment. The Build Tool does not interact with Pathway and therefore cannot create the Guardian target environment as part of the build process. However, the Build Tool generates several files that aid in creating the Guardian target environment. The Build Tool tokens OPT.NS_SERVER_TARGET_VOLUME and OPT.NS_SHORT_MODEL_NAME represent the Volume and Subvolume of the Guardian target environment.

Note: When creating cooperative servers, many of the steps that are taken to create the Guardian target environment are not needed. Those required for the BlockMode applications are indicated.

The following steps are taken to create the Guardian target environment:

Create the SQL/MP Catalog (BlockMode only)

Use sqlci to create the catalog. Use the same values given with Build Tool tokens OPT.NS_SERVER_TARGET_VOLUME and OPT.NS_SERVER_SHORT_MODEL_NAME for the volume and subvolume names.

FTP Generated Pathway Files

Several Pathway files are generated as part of the application build using the Build Tool. The following list of files are FTP'd to the Guardian target environment:

- AEF
- *.pwy
- DROPTBLS (BlockMode only)
- FUPFILES
- PATHCNFG
- PWCOLD
- PWCOOL
- PWSTOP
- SQLTMT (BlockMode only)
- SRVRLOG
- SRVRMSG (BlockMode only)
- SRVROBEY

- SRVRREC (BlockMode only)
- SRVREPL (BlockMode only)
- SRVRTMT (Cooperative only)
- TMTE
- ZTMT (BlockMode only)

If FTP’ing manually, ensure that the *.pwy files are renamed in the Guardian target environment without the .pwy file extension.

Optionally, an additional TACL macro FTPFILES has been generated that aids in FTP’ing these files. This file is modified to provide the hostname, userid, and password. Manually FTP’ing this file to the Guardian target environment and executing will FTP the list of files automatically.s

Copy Additional Guardian Pathway Files

The CA Gen IT delivers additional files are required to run the generated application. The following list of files are copied to the Guardian target environment:

- LOGSERV
- MSGRT (BlockMode only)
- POBJCOD
- POBJDIR
- POBJSYM
- RECSERV (BlockMode only)
- REPLAY (BlockMode only)
- DPSTMTC (Cooperative only)

The generated file FUPFILES was FTP’d and can be obeyed to copy these files to the Guardian target environment.

Create and Populate Transaction Mapping Table (BlockMode only)

To run the AEF to test/run the generated application, the AEF uses a table that is created and populated. The generated file SQLTMT can be executed that performs the following steps:

- Add defines for the TMT and RECAP tables
- Create the TMT and RECAP tables
- SQLCOMP the MSGRT and REPLAY servers
- Populate the TMT table

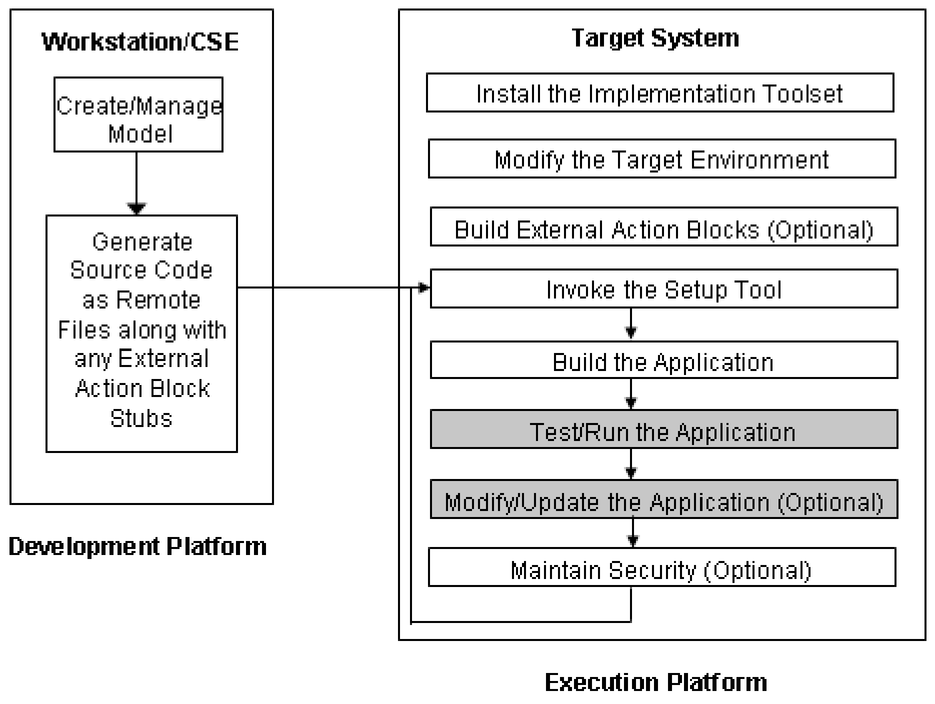
The file SQLTMT has been generated to assist with these steps. This file should have been FTP’d to the Guardian target environment, and can be run to execute the previous steps.

Now, that the target environment has been created, the application can be tested, run, or both.

Chapter 7: Testing and Running Applications in NonStop

Although an application is ready for execution after the remote files are processed, we recommend that you test your application using the Diagram Trace Utility before introducing the application into your production environment.

The relationship of testing and running your application to the other required and optional IT tasks is described in the following illustration:



Workstation/CSE
Create/Manage Model
Generate Source Code as Remote Files along with any External Action Block Stubs
Development Platform
Target System
Install the Implementation Toolset
Modify the Target Environment
Build External Action Blocks (Optional)
Invoke the Setup Tool
Build the Application
Test/Run the Application
Modify/Update the Application (Optional)
Maintain Security (Optional)
Execution Platform

Load an Application

After an application is built successfully, it is ready to run. Running an application starts the application servers in the Pathway environment. After the application servers are loaded, they can be accessed through the AEF, the non-TCP Requester, or RSC/MP (for Distributed Servers).

Follow these steps:

1. Set your current volume to the application's target location.

   ```
   VOLUME $<setvol>.<setsubvol>
   ```

2. Cold-load your application's Pathway environment.

   ```
   PWCOLD
   ```

   This command cold-loads a newly created Pathway environment. The application starts running.

Stop an Application

After you finish running the application, you can stop it.

Follow these steps:

1. Set your current volume to the application's target location.

   ```
   VOLUME $<setvol>.<setsubvol>
   ```

2. Stop the application by issuing the PWSTOP command.

   The application stops running.

Application Startup Parameters (Block Mode only)

For Block Mode applications, you do not have to enter information directly on the screen as the startup parameters supply information to the executable procedure. The executable procedure must be an online procedure and must be defined as one that accepts clear screen input. Clear screen input is defined with the Dialog Flow Diagramming Tool.

Clear screen input can be accepted by both the AEF and the non-TCP Requester. The clear screen input consists of one or more parameters separated with the appropriate string and parameter delimiters. You can define the string and parameter delimiters through Business System Defaults.

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

The parameter list is defined by keywords. Keywords are labels that identify the parameter and must be accompanied by the parameter data enclosed in delimiters.

The format for parameters is:

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

KEYWORD1, KEYWORD2

Defines the keywords.

<space>

Identifies the keyword separator or data separator.

data1, data2

Defines the data associated with the keyword to be used by the executable procedure.

<comma>

Identifies the 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. All keywords must be unique.

When using keywords, the parameters entered need not be in the same order as the parameter list defined for the screen. For example, the following two clear screen inputs are equally acceptable:

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

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

Keywords are case-sensitive and cannot contain delimiters. However, you can append symbols. For example, you can append an equal sign (=) to the keyword NAME. The result 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 support the RESTART parameter.

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

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

Non-TCP requesters can supply additional initial data to application servers.

More information:

Interfacing with a Non-TCP Requester (see page 79)

AEF Overview

The Application Execution Facility (AEF) is an environment that lets you execute CA Gen Block Mode applications on NonStop. The AEF is tightly based on Pathway and provides the following functionality:

- Error handling
- Standard runtime library (providing access to existing applications)
- Dialog flow
- Terminal mapping

In addition, the Pathway architecture provides the following functionality:

- Transaction control
- Fault tolerance
- Transaction recovery

Invoke AEF

You can access AEF using the following devices:

- NonStop 6530-type terminals
- IBM 3270-type terminals

NonStop 6530-type terminals can be attached to the AEF using the following transport protocols:

- Asynchronous lines
- TCP/IP

Terminal emulators that support the 6530 block-mode protocol are also supported.

Since Pathway supports IBM 3270-type devices, you can connect such devices (or compatibles) to the AEF using the following access methods:

- AM3270
- SNAX/XF
- SNAX/CDF

IBM 3270-type devices must be secondary devices on the line. The NonStop server must be the primary device through the SNAX line.

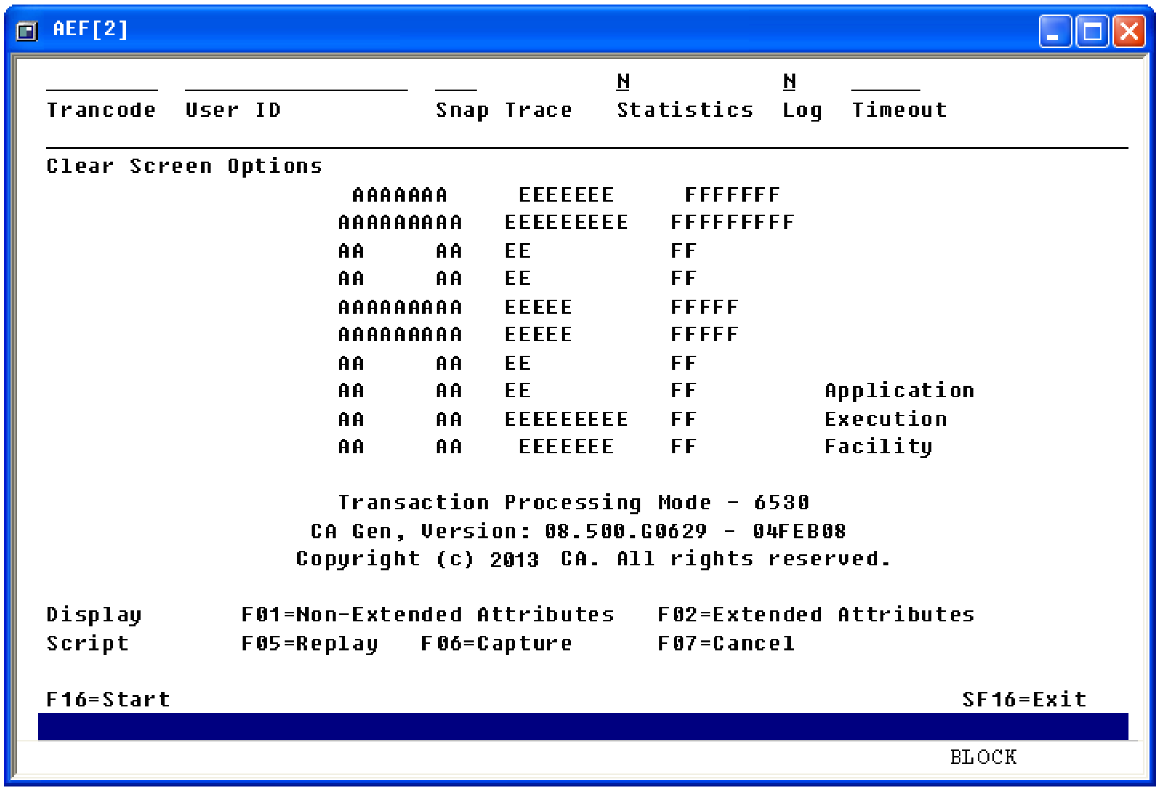
Follow these steps:

1. Set your current volume to the application's target location.

   ```
   VOLUME $<setvol>.<setsubvol>
   ```

2. Start the AEF by issuing the AEF command.

   The AEF screen appears.



```
AEF[2]

                                                 N           N
________   ________________   ___                            _____
Trancode   User ID            Snap Trace   Statistics   Log   Timeout
_______________________________________________________________________________
Clear Screen Options
                        AAAAAAA     EEEEEEE     FFFFFFF
                       AAAAAAAAA   EEEEEEEEE   FFFFFFFFF
                       AA      AA   EE          FF
                       AA      AA   EE          FF
                       AAAAAAAAA   EEEEE       FFFFF
                       AAAAAAAAA   EEEEE       FFFFF
                       AA      AA   EE          FF
                       AA      AA   EE          FF          Application
                       AA      AA   EEEEEEEEE   FF          Execution
                       AA      AA    EEEEEEE    FF          Facility

                       Transaction Processing Mode - 6530
                     CA Gen, Version: 08.500.G0629 - 04FEB08
                    Copyright (c) 2013  CA. All rights reserved.

Display       F01=Non-Extended Attributes   F02=Extended Attributes
Script        F05=Replay    F06=Capture      F07=Cancel

F16=Start                                                        SF16=Exit
                                                              BLOCK
```

The AEF screen contains the following input fields:

Trancode

The trancode with which the application will start. This is a required field.

User ID

(Optional) Overrides the default user ID for the current AEF session.

Snap Trace

(Optional) Specifies the level of information logged to an internal trace file. This log file is created in the base target subvolume and is named lg*xxxxxx* (where *xxxxxx* is the process ID of the running server). A value between 0 and 31 is provided—the higher the value, the more information will be traced. A trace value of 31 may consume a large amount of disk space, which can significantly slow processing.

Note: Use of the Snap Trace field should be for development purposes and only under the direction of Technical Support to locate problems.

Statistics

(Optional) Indicates to collect transaction statistics and record them to the STATIS file located in the base target subvolume.

Values: Y or N

Default: N

Log

(Optional) Indicates whether the AEF Requester will log all encountered error conditions to the EMS log.

Values: Y or N

Default: N

Timeout

(Optional) Specifies the amount of time given in seconds before the application is terminated.

Default: No default value

Clear Screen Options

Identifies the input supplied to the executable procedure.

Note: For more information, see Application Startup Parameters (Block Mode only) (see page 62).

3. Press the F2 function key followed by the F16 function key to invoke the associated load module for execution after the transaction code and optional fields are populated. This effectively starts the application.

   Upon exiting from the application, the load module terminates and control returns to the AEF.

Only Block Mode applications built as transactional are invoked using the AEF. Command line applications are invoked through a non-TCP requester.

Note: Function key assignments for the AEF may differ for each terminal device.

The AEF uses a keyboard mapping file to assign values to each function key and special purpose key available for use by any generated applications that reside on your target system. The following files are available for use:

- T3270ES
- T3270NS
- T6530ES
- T6530NS

More information:

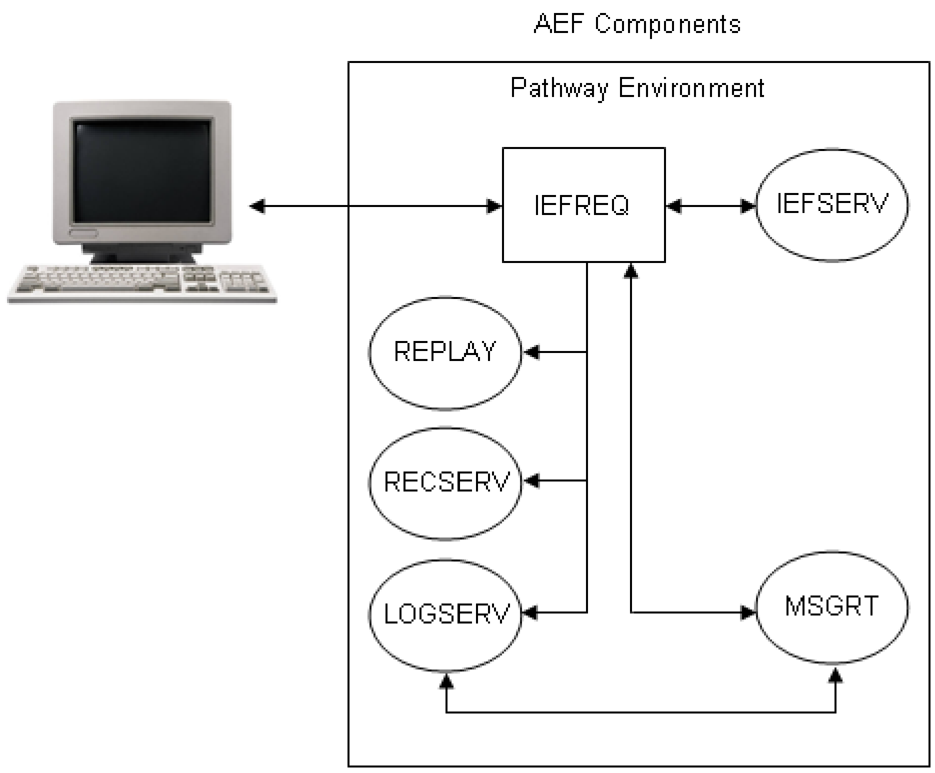
Application Startup Parameters (Block Mode only) (see page 62)
How to Map a Function Key to the Exit Key Defined for a E107 Application (see page 103)

AEF Components

The following components are available in AEF:

- AEF Requester (IEFREQ)
- AEF Server Interface (IEFSERV)
- Message Router Server (MSGRT)
- Error Log Server (LOGSERV)
- Replay Server (REPLAY)
- Statistics Server (RECSERV)

The following illustration shows the interaction of the AEF components:



The AEF components are described in the following sections.

AEF Requester

The AEF Requester (IEFREQ) is a program that runs through the Terminal Control Process (TCP) and implements the following functions:

- Terminal reading
- Terminal writing
- Message delivery
- Message switching
- AEF configuration start-up

- Transaction control
- Checkpointing

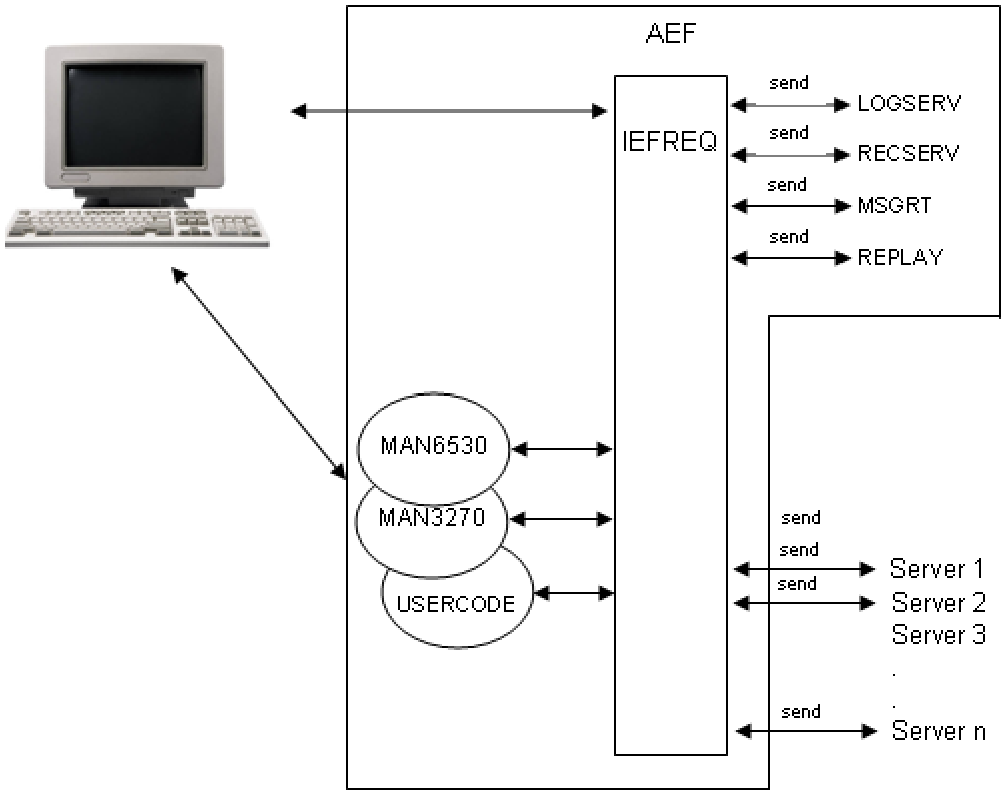
IEFREQ is responsible for routing incoming transactions to the appropriate application server class. It can communicate with the following servers:

- Application Server (CA Gen generated)
- Message Router Server
- Error Log Server
- Replay Server
- Statistics Server

IEFREQ is structured so that it can be incorporated into an existing Pathway application. It logs error conditions to the Event Management Subsystem (EMS) through the Error Log Server (LOGSERV).

The IEFREQ program must be the terminating SCOBOL program in a calling sequence; that is, the IEFREQ program does not support calling child SCOBOL programs.

The following illustration shows how IEFREQ interfaces with user-written requesters, application servers, and the AEF supported servers:



AEF Server Interface

To understand the role of the AEF Server Interface (IEFSERV), this section describes a CA Gen generated server, of which IEFSERV is a part.

The CA Gen generated server is comprised of three parts:

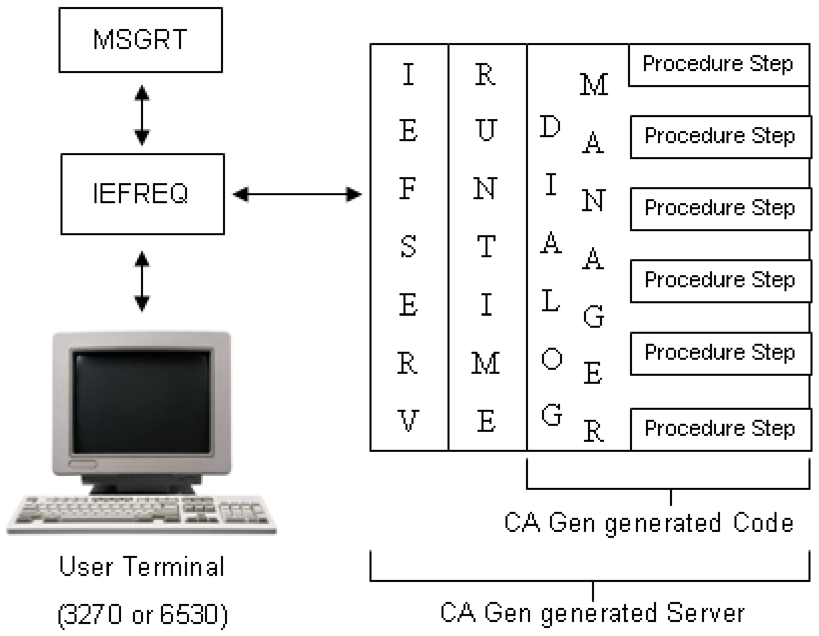
- AEF Server Interface
- CA Gen runtime
- CA Gen generated code

The IEFSERV is bound to the CA Gen runtime and the CA Gen generated code to create an application server. The application server is a server class in the Pathway environment. A different server class is generated based on the load module packaging that occurs within the Business System design.

The IEFSERV and the CA Gen runtime are delivered as part of the Implementation Toolset while you develop the CA Gen generated code. All three components are linked together during the build of an application by the Setup Tool.

IEFSERV is a collection of routines that reads and replays messages through $RECEIVE, handles all the CA Gen intrinsic functions, handles the overflow file, and reports errors to LOGSERV.

The following illustration provides the basic structure of the CA Gen generated server:



Message Router Server

The Message Router Server (MSGRT) is a server that implements Transaction Mapping functions.

On initial startup, MSGRT loads the Transaction Mapping Table (TMT) into extended memory. This improves the performance of the binary search that the message router uses to find a transaction code. If a change in the CA Gen procedure step packaging is required, the message routers must be stopped.

MSGRT locates the incoming transaction code from the input data stream, and performs a table lookup (from the TMT) to map the transaction code to the CA Gen server class responsible for handling that business function.

The TMT is defined to MSGRT as a define called =TMT. The definition of =TMT is included in the default DEFINES file created during the build of an application using the Setup Tool.

The TMT holds a set of transaction codes associated with executable load modules. This table resides in the application's target location. There is one TMT for each application. During application execution, this table is read into memory for performing lookups to ascertain the appropriate server that handles a particular transaction.

Each transaction code must be unique within each TMT. If the transaction codes are not unique, you may encounter problems during load module execution. The TMT can store up to 1024 transaction codes.

The TMT is comprised of the following columns:

```
TID                CHAR(8) NO DEFAULT NOT NULL
SERVER_CLASS_NAME  CHAR(15) DEFAULT NULL
MODEL_NAME         CHAR(32) DEFAULT NULL
```

Example:

```
> volume $VDATA3.ABLK11
> sqlci
>> select * from tmt;
TID       SERVER_CLASS_NAME   MODEL_NAME
--------  -----------------   ----------------------
P130      P130                PT ABLK TEST MODEL 11
P131      P131                PT ABLK TEST MODEL 11
P132      P132                PT ABLK TEST MODEL 11
P133      P133                PT ABLK TEST MODEL 11
P134      P134                PT ABLK TEST MODEL 11
--- 5 row(s) selected.
>> exit
```

Error Log Server

The Error Log Server (LOGSERV) is a server that implements the error logging function.

LOGSERV is the CA Gen gateway to the EMS software running on NonStop. LOGSERV subsequently formats the message into a tokenized message for routing to an EMS collector ($0 by default) through the EMS subsystem. Events can originate from IEFSERV, IEFREQ, or MSGRT.

CA Gen system messages are visible with operation management products such as ViewPoint.

Replay Server

The AEF can record and playback a session with an application. You can play the recording at a later time only if the database associated with the application has not been changed.

The Replay Server (REPLAY) records and replays transactions. These transactions are stored in the SQL table named RECAP. The RECAP table is defined to REPLAY as a define called =RECAP. The definition of =RECAP is included in the default DEFINES file created during the build of an application using the Setup Tool.

Perform the following steps to record transactions.

Follow these steps:

1. Start the AEF.
2. Enter a valid transaction code in the Trancode field.
3. Press the F1 or F2 function key to establish the terminal type.
4. Press the F6 function key to enable the capture of transactions.

   An overlay will be displayed at the bottom of the screen.
5. Enter the capture ID name that uniquely identifies this session.
6. Type **Y** or **N** in the Append field. If you are using an existing capture ID name, you have the option of appending a new session to it by entering **Y**. **N** overrides existing sessions with the same ID.
7. Press the SF16 function key to exit the overlay.
8. Press the F16 function key to start capturing, or press the F7 function key to quit.

   The transaction is recorded.

Perform the following steps to replay a recorded session.

Follow these steps:

1. Start the AEF.
2. Enter the same transaction code you entered in the Trancode field when recording transaction.
3. Press the F1 or F2 function key to establish the terminal type.
4. Press the F5 function key to access the Replay Configuration overlay.
5. Enter the capture ID that identifies the session you want to replay.
6. Enter the delay time in seconds to determine the amount of time that lapses between screens. The default value is 0.
7. Enter the sequence number. Each recorded transaction is assigned an incremental sequence number in the RECAP table. Recorded transactions begin with sequence number 0. You can enter any sequence number as long as the associated recorded transaction is applicable to the transaction code that you have entered at the top of the AEF. The default value is 0.
8. Press the SF16 function key to exit the overlay.
9. Press the F16 function key.

   The recorded transaction is replayed.

The RECAP table is comprised of the following columns:

```
USER_ID          PIC X(16) NO DEFAULT NOT NULL
CAPTURE_ID       PIC X(8) NO DEFAULT NOT NULL
SEQ_NUM          PIC 9(5) COMP NO DEFAULT NOT NULL
TERMINAL_TYPE    PIC 9(4) COMP NO DEFAULT NOT NULL
TERMINAL_SUBYPE  PIC 9(4) COMP NO DEFAULT NOT NULL
DATA_LENGTH      PIC 9(5) COMP NO DEFAULT NOT NULL
TERM_DATA        PIC X(4000) DEFAULT NULL
```

Example:

```
> volume $VDATA3.ABLK11
> sqlci
>> select * from recap;

USER_ID          CAPTURE_ID  SEQ_NUM    TERMINAL_TYPE  TERMINAL_SUBYPE  DATA_LENGTH
---------------- ---------- ---------- ------------- --------------- -----------
TERM_DATA
----------------------------------------------------------------------------

002-ZN018-000     FRED                0             49              50          26
V!(/P120P120S5

002-ZN018-000     FRED                1             49              50          23
B!  P123P123SY
002-ZN018-000     FRED                2             49              50          26
V!(/P120P120S4
002-ZN018-000     FRED                3             49              50          23
B!"!P122P122SY
002-ZN018-000     FRED                4             49              50          26
V!(/P120P120S6
002-ZN018-000     FRED                5             49              50          71
V!2:P124P124SY 1111111111111111111+ 44444444444444444

002-ZN018-000     FRED                6             49              50          23
B!&#P124P124SY
--- 7 row(s) selected.
>> exit
```

Statistics Server

The Statistics Server (RECSERV) is a Pathway server that implements the Time Recording function. It collects transaction statistics and records them in the STATIS file.

RECSERV is activated by IEFREQ through flag settings in the configuration parameters (set the Statistics flag on the AEF Main Menu to Y). RECSERV logs errors to LOGSERV.

The STATIS file is comprised of the following fields:

```
Transaction ID     char[9]
Initial Time       char[9]
Ending Time        char[9]
Bytes to Server    char[6]
Bytes from Server  char[6]
Terminal Name      char[17]
Server Name        char[16]
```

Example:

```
P530     11044293 _11044299 _00078 _01099 _002-ZN018-000 P530
P532     11045051 _11045059 _00027 _01685 _002-ZN018-000 P530
P532     11045725 _11045732 _00031 _00917 _002-ZN018-000 P530
P530     11050129 _11050133 _00026 _01099 _002-ZN018-000 P530
```

This file is an editable file (file code 101).

Application Testing

The following sections describe application testing and the procedure required to access and use the Diagram Trace Utility.

Diagram Trace Utility

The Diagram Trace Utility is used to test a generated application before it is moved to a production environment.

The Diagram Trace Utility provides a number of special capabilities that allow you to view CA Gen model elements used to build an application (such as Action Diagram statements) while the program is being executed. To use the special capabilities of the Diagram Trace Utility, you need to make certain selections when remote files are generated. These selections cause the generation of the additional code required to implement the special capabilities available through the Diagram Trace Utility.

The portions of the application that are to be tested must be generated with trace enabled to communicate with the Diagram Trace Utility.

Before an application can be tested, certain application components must be built using the Setup Tool:

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

Enable the Diagram Trace Utility

You must use the Diagram Trace Utility to test a generated application.

Note: Ensure that NonStop TCP/IP is installed and running for TCP/IP connectivity before using the Diagram Trace Utility.

Follow these steps:

1. Build the test application that has been generated with trace turned on.
2. Invoke the Diagram Trace Utility on any accessible Windows system:
   - Launch the Diagram Trace Utility from the Start menu.
   - Click Start, All Programs, CA, Gen *xx*, Diagram Trace Utility.

     xx is the current CA Gen version installed on your system.

     The Diagram Trace Utility automatically starts listening on port 4567. You can change the default port in the Diagram Trace Utility.

     Note: For more information about changing the default port, see the *Diagram Trace Utility User Guide*.
3. Set the trace environment variables in the applini file. This file is delivered with the Implementation Toolset and by default, it is placed in the same directory or subvolume as the executables (<base-location>X) during the deployment or installation of CA Gen applications.
4. Edit the Pathway server configuration files for those load modules (servers) that you intend to debug or trace. By default, the Pathway configuration servers will be found in the <base-location>U directory or subvolume. To support the Diagram Trace Utility, the following DEFINE entries exist in each file:
   - =APPLICATION_INI—is the fully qualified name of the application initialization file
   - =TCPIP^PROCESS^NAME—is the name of the TCP/IP process running the Listener and Telnet processes on the system.

     Default: $ZTC0

     Note: There may be more than one TCP/IP process running on a system. Contact your system administrator to find out the name of this process.

     All servers must have the same =APPLICATION_INI and =TCPIP^PROCESS^NAME DEFINES settings.

     Example: The following example is a sample of the proposed DEFINES:

     SET SERVER DEFINE =APPLICATION_INI,CLASS MAP,FILE $DATA1.PROXYX.APPLINI

     SET SERVER DEFINE =TCPIP^PROCESS^NAME,CLASS MAP,FILE $ZB018
5. After making the appropriate changes to the application initialization file and the Pathway server configuration files, start your applications using the PWCOLD command.

   Note: PWCOOL will not always work because of the changes made to the Pathway server configuration files.

   The Diagram Trace Utility is ready for use.

More information:

Environment Variables in the Applini File (see page 129)

Access and Use the Diagram Trace Utility

You can test all the generated C applications using the Diagram Trace Utility on NonStop. Each application requires transaction codes to map to the correct Procedure Steps and Action Blocks.

After you enable the Diagram Trace Utility, invoke your application based on the information provided in the following table:

C Application	Invocation or Access Method	How Trancode is Obtained
Distributed Processing Server	Pathway through RSC/MP	Transaction Mapping Table
Block Mode Commandline Nodisplay	Pathway through non-TCP Requester	Transaction Mapping Table
Block Mode Interactive	Pathway through AEF	Transaction Mapping Table

While testing an application, any of its procedure steps or action blocks that have been generated with trace will communicate with the Diagram Trace Utility. The application screens (Block Mode only) will be updated after control is returned from the Diagram Trace Utility.

The Diagram Trace Utility lets you step through the execution of the application by viewing the sequence of action block calls and action diagram statements being executed as the generated program executes.

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

Multi-user Diagram Trace Utility Support (Distributed Server Applications Only)

When a remote server application is started, one copy of the applini file for that application is available. Setting the trace environment variables within this applini file will allow only one Diagram Trace Utility to trace any of the servers that make up this server application.

Test environments involve multiple testers working with the same server application. In this scenario, you need to debug from multiple client workstations and therefore, use multiple Diagram Trace Utilities.

More information:

Using the Applini File (see page 129)

Override the Default Trace Environment Variables

A client transaction can transfer its host and port information of the client that is executing it. By providing this information, the server can establish communication with the Diagram Trace Utility running on that client workstation.

Follow these steps:

1. Build the server application that has been generated with trace (as before).
2. Leave the trace environment variables within the applini file that is located in the server application's model directory commented out.
3. Start your servers using the PWCOLD command.
4. Invoke the Diagram Trace Utility on your client workstation.
5. Edit the application.ini file residing in your client application's executables directory.
6. Uncomment and set the trace environment variables. You can use *localhost* for the TRACE_HOST variable.
7. Launch the Client Manager on your client workstation and configure to communicate with your server.
8. Execute your client application.

   The Diagram Trace Utility on the client workstation will be used to trace the server for each transaction initiated from that client.

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.

Chapter 8: Interfacing with a Non-TCP Requester

Interfacing with a Non-Terminal Control Process Requester With an IPCO Library

A non-TCP requester is an entity program that resides outside the boundaries of the AEF. As a result, the Terminal Control Process (TCP) that usually monitors all terminal activity generated by traditional requesters has no control over the execution of non-TCP requesters.

For a non-TCP requester to establish communication with the AEF environment, a set of routines is developed to form an interface. This interface is delivered as a library named IPCO. It can be found in the subvolume where the Implementation Toolset is located. The IPCO interface allows a user-written non-TCP requester to communicate to any of the CA Gen application servers that typically do not contain any user interface (like the Blockmode Commandline Nodisplay applications).

The IPCO library contains the following routines:

- INITIALIZE^IPC—Sets up internal data structures
- DECODE^MSGRT^IPC—Verifies call to MSGRT server
- DECODE^SERVER^IPC—Decodes information from IEFSRV
- PREPARE^SERVER^IPC—Set attributes within AEF data structures

The following sections provide more information about each of these routines.

Note: Two sample programs are provided to assist you in understanding the use of the procedure calls. You can find the source in the NTCPR1S and NTCPR2S files that are available in the subvolume where the IT is located.

INITIALIZE^IPC Routine—Set Up Internal Data Structures

The non-TCP requester calls the INITIALIZE^IPC routine once before calling any of the other procedures. This routine sets up internal data structures that will be used to communicate with the application servers.

Usage

The syntax of the INITIALIZE^IPC call is:

```
CALL INITIALIZE^IPC     (trancode                      !input
                           ,options                       !input
                           ,display^terminal              !input
                           ,display^terminal^type         !input
                           ,display^terminal^subtype      !input
                           ,debug^terminal                !input
                           ,debug^terminal^type           !input
                           ,debug^terminal^subtype        !input
                           ,userid                        !input
                           ,printer^name                  !input
                           ,flag^switch^message           !input
                           ,flag^test                     !input
                           ,flag^trace                    !input
                           ,trace^level                   !input
                           ,MAX^REPLY^LENGTH              !input
                           ,ipc^interface                 !output
                           ,write^length                  !output
                           ,error^code                    !output
                           ,error^subcode );              !output
```

Description

The INITIALIZE^IPC routine performs the following tasks.

- Verify a valid terminal type
- Verify a valid user's buffer size
- Compute the length of the buffer used later by SERVERCLASS_SEND

After the call to INITIALIZE^IPC routine is complete, a call to MSGRT must follow. This call must use the IPC^INTERFACE buffer to pass all the information necessary to obtain the name of the server class that will satisfy the initial request.

Parameters

trancode

I/O: Input

STRING:ref:*

Identifies the transaction ID used to start the application. Only the first eight bytes of the ID are used to populate the transaction code in the internal data structure.

options

I/O: Input

STRING:ref:*

Defines clear screen options such as RESET or RESTART. The options field can be up to 80 bytes in length, which can include spaces.

display^terminal

I/O: Input

INT:ref:*

Identifies the name of the terminal in external format. Only local terminal names of up to 16 bytes in length are allowed.

display^terminal^type

I/O: Input

STRING:ref:*

Identifies a one-byte field denoting the terminal type. The following values are valid for this field.

1

Indicates 6530 terminal types.

2

Indicates 3270 terminal types.

display^terminal^subtype

I/O: Input

INT:ref:*

Identifies a one-byte field denoting the terminal attributes. The following values are valid for this field.

1

Indicates non-extended attributes.

2

Indicates extended attributes.

debug^terminal

I/O: Input

STRING:ref:*

Identifies a one-byte input value that is no longer used.

debug^terminal^type

I/O: Input

INT:ref:*

Identifies a one-byte input value that is no longer used. Must be set to 0.

debug^terminal^subtype

I/O: Input

INT:ref:*

Identifies a one-byte input value that is no longer used. Must be set to 0.

userid

I/O: Input

STRING:ref:*

Identifies the user running the application. Values up to 16 bytes in length are allowed.

printer^name

I/O: Input

STRING:ref:*

Identifies the name of the printing device. Values up to 36 bytes in length are allowed.

flag^switch^message

I/O: Input

INT:ref:*

Identifies a one-byte input value that is no longer used. Must be set to 0.

flag^test

I/O: Input

INT:ref:*

Identifies a one-byte input value that is no longer used. Must be set to 0.

flag^trace

I/O: Input

INT:ref:*

Identifies a one-byte input value that is no longer used. Must be set to 0.

trace^level

I/O: Input

INT:value

Defines an input value that exists for internal use only. Must be set to 0.

max^reply^length

I/O: Input

INT:value

Identifies the size of the buffer interface. The minimum size of this field must be equal to or greater than 8 K bytes and less than or equal to 32 K bytes.

ipc^interface

I/O: Output

STRING.EXT:ref:*

Identifies a field that contains all the configuration parameters that are to be forwarded to the application servers.

write^length

I/O: Output

INT:ref:*

Identifies a field that returns the length of the buffer being sent back to the calling procedure.

error^code

I/O: Output

INT:ref:*

Identifies a field that returns a value showing the state of the call after it completes.

1

Indicates an invalid buffer size (less than 8 K).

2

Indicates an invalid terminal type.

-1

Indicates no errors.

error^subcode

I/O: Output

INT:ref:*

Identifies a field always returning a value of -1.

Example

```
! set the initial information !
   CALL get^initial^info( pathmon^name
                         ,pathmon^name^len
                         ,trancode
                         ,options
                         ,display^terminal
                         ,display^terminal^type
                         ,display^terminal^subtype
                         ,flag^test
                         ,debug^terminal
                         ,debug^terminal^type
                         ,debug^terminal^subtype
                         ,flag^trace
                         ,trace^level
                         ,flag^switch^message
                         ,userid
                         ,printer^name );
! Initialize IPC info
   CALL INITIALIZE^IPC( trancode
                       ,options
                       ,display^terminal
                       ,display^terminal^type
                       ,display^terminal^subtype
                       ,debug^terminal
                       ,debug^terminal^type
                       ,debug^terminal^subtype
                       ,userid
                       ,printer^name
                       ,flag^switch^message
                       ,flag^test
                       ,flag^trace
                       ,trace^level
                       ,MAX^REPLY^LENGTH
                       ,buffer
                       ,write^length
                       ,error^code
                       ,error^subcode );
! Check for errors
  IF error^code <> -1 then
            call error^routine;
```

DECODE^MSGRT^IPC Routine—Verify Call to MSGRT Server

Use the DECODE^MSGRT^IPC routine after each call to MSGRT. You can use this routine to obtain information about the server class name responsible for the execution of the current transaction ID.

Usage

The syntax of the DECODE^MSGRT^IPC call is:

```
CALL DECODE^MSGRT^IPC( ipc^interface                 !input
                      ,reply^code                    !output
                      ,trancode                      !output
                      ,trancode^len                  !output
                      ,server^class^name             !output
                      ,server^class^len              !output
                      ,write^length                  !output
                      ,error^code                    !output
                      ,error^subcode );              !output
```

Description

The DECODE^MSGRT^IPC routine verifies if the call to the MSGRT server is successful. It also obtains routing information through the transaction ID and the server class name parameters.

Parameters

ipc^interface

I/O: Input

STRING.EXT:ref:*

Identifies an interface buffer that contains all the configuration parameters, which are forwarded to the MSGRT server.

reply^code

I/O: Output

INT:ref:*

Indicates the output value.

0

Indicates no error occurred.

1

Indicates an invalid terminal type was passed to the MSGRT within the interface buffer.

trancode

I/O: Output

STRING:ref:*

Identifies a value representing the transaction ID from the TMT table.

trancode^len

I/O: Output

INT:ref:*

Indicates the length (in bytes) of the transaction ID returned in the trancode.

server^class^name

I/O: Output

STRING:ref:*

Identifies the name of the server class found in the TMT table that satisfies the incoming transaction ID.

server^class^length

I/O: Output

INT:ref:*

Identifies the length (in bytes) of the server class name.

pathsend^write^length

I/O: Output

INT:ref:*

Identifies the length of the data returned by MSGRT.

error^code

I/O: Output

INT:ref:*

Indicates a value showing the state of the call when it completes.

-1

Indicates no errors.

4

Indicates that an invalid transaction code was sent to the MSGRT server.

error^subcode

I/O: Output

INT:ref:*

Identifies a field always returning a value of -1.

Example

```
!Send the data buffer to the MSGRT server using the SENDSERVER_CLASS_call
server^class^name':='"MSGRT";
server^class^name^len:=5;
error := SERVERCLASS_SEND_( pathmon^name
                           ,pathmon^name^len
                           ,server^class^name
                           ,server^class^name^len
                           ,data^buffer
                           ,write^length
                           ,MAX^REPLY^LENGTH
                           ,count^read
                           ,timeout
                           ,flags
                           ,^ScSendOpNum );
IF error <> FEOK THEN
call error^routine;
CALL DECODE^MSGRT^IPC( data^buffer
                      ,reply^code
                      ,trancode
                      ,trancode^len
                      ,server^class^name
                      ,server^class^name^len
                      ,write^len
                      ,error^code
                      ,error^subcode );
```

```
! Check for errors
  IF error^code <> -1 then
            call error^routine;
```

DECODE^SERVER^IPC Routine—Decode Information from IEFSRV

The user program must call the IEF^DECODE^SERVER^IPC routine after every call to IEFSRV. This routine decodes (translates) information from IEFSRV and writes it to a buffer.

Usage

The syntax of the DECODE^SERVER^IPC call is:

```
CALL DECODE^SERVER^IPC( ipc^interface               !input
                       ,reply^code                  !output
                       ,flag^change                 !output
                       ,flag^abort^transaction      !output
                       ,server^class^name           !output
                       ,server^class^len            !output
                       ,data^message                !output
                       ,data^length                 !output
                       ,diagnostic^message          !output
                       ,diagnostic^length           !output
                       ,write^length                !output
                       ,error^code                  !output
                       ,error^subcode );            !output
```

Description

A non-zero value in the reply^code parameter can imply any one of the following:

- An invalid terminal type was passed to the application server in the interface buffer.
- There was an error during an I/O operation involving the overflow file.

The IEFREQ requester has the limitation of handling only up to 8 K of data. When this happens, the source server writes the remaining data to a temporary file called the overflow file. An overflow file is used when a server is unable to pass the entire data buffer back to the requester. The server then passes the information back to the requester including the name of this temporary file. The IEFREQ requester forwards this information to the target server, which also reads the overflow file to obtain the remaining information.

A value of 0 in the flag^change parameter implies that a reply should be forwarded to you. In that case, the data^message buffer will receive the data returned from the server. The data^length parameter will have the length, in bytes, of this buffer. If a diagnostic message is present, it will be placed in the diagnostic^message buffer. Diagnostic^length has the length of the diagnostic^message buffer.

If the flag^change parameter is set to 1, then the server^class^name will have the name of the server to which the reply should be forwarded. Server^class^length has the length, in bytes, of the name of the server, and pathsend^write^length contains the size of the data that should be passed to the next server class.

Also, the flag^abort^transaction will be set if the server detects a problem.

Parameters

ipc^interface

I/O: Input

STRING.EXE:ref:*

Identifies a buffer interface that receives data returned from the application server.

reply^code

I/O: Output

INT:ref:*

Returns 0 if no error occurs during the call.

flag^change

I/O: Output

INT:ref:*

1

Indicates that a dialog flow will occur to another server class. In this case, server^class^name contains the name of the new server class.

0

Indicates that no dialog flow will take place.

flag^abort^transaction

I/O: Output

INT:ref:*

0

Indicates the calling requester program to finish the current transaction normally (END-TRANSACTION)

1

Indicates the calling requester program to abort the current transaction (ABORT-TRANSACTION)

server^class^name

I/O: Output

STRING:ref:*

Identifies the name of the server class to which the data will be forwarded.

server^class^length

I/O: Output

INT:ref:*

Identifies the length (in bytes) of the server class name.

data^message

I/O: Output

STRING.EXT:ref:*

Identifies a pointer to the buffer containing the data returned from the applications servers.

data^length

I/O: Output

INT(32):ref:*

Identifies the length of the data^message buffer.

diagnostic^message

I/O: Output

STRING.EXT:ref:*

Identifies a pointer to the diagnostic message buffer.

diagnostic^length

I/O: Output

INT:ref:*

Identifies the length (in bytes) of the diagnostic message buffer.

pathsend^write^length

I/O: Output

INT:ref:*

Returns the length (in bytes) of the buffer to be sent to the application server.

error^code

I/O: Output

INT:ref:*

Identifies a field always returning a value of -1.

error^subcode

I/O: Output

INT:ref:*

Identifies a field always returning a value of -1.

Example

```
! Send the data buffer to the server denoted by the server^class^name parameter
! using the SENDSERVER_CLASS_call
error := SERVERCLASS_SEND_( pathmon^name
                           ,pathmon^name^len
                           ,server^class^name
                           ,server^class^name^len
                           ,data^buffer
                           ,write^length
                           ,MAX^REPLY^LEN
                           ,count^read
                           ,timeout
                           ,flags
                           ,^ScSendOpNum );
IF error^code <> FEOK then
            call error^routine;
```

```
!Verify SERVER information
CALL DECODE^SERVER^IPC( data^buffer
                       ,reply^code
                       ,flag^change
                       ,flag^abort^transaction
                       ,server^class^name
                       ,server^class^name^len
                       ,data^message
                       ,data^length
                       ,diagnostic^message
                       ,diagnostic^length
                       ,write^length
                       ,error^code
                       ,error^subcode );
! Check for errors
  IF error^code <> -1 then
            call error^routine;
```

PREPARE^SERVER^IPC Routine—Set Attributes Within AEF Data Structures

The user program must call this procedure each time the requester is ready to prepare the data buffer to be sent to the application servers.

Usage

The syntax of the PREPARE^SERVER^IPC call is:

```
CALL PREPARE^SERVER^IPC( data^message                !input
                        ,data^length                 !input
                        ,trancode                    !input
                        ,trancode^len                !input
                        ,ipc^interface               !input/output
                        ,write^length                !output
                        ,error^code                  !output
                        ,error^subcode );            !output
```

Description

You can use the PREPARE^SERVER^IPC routine to set the appropriate attributes within the AEF data structures. It must be called before sending data to the application servers.

Parameters

data^message

I/O: Input

STRING.EXT:ref:*

Identifies a pointer to the buffer containing the data to be passed to the application.

data^length

I/O: Input

INT(32) value

Identifies a field that contains the length of the data^message buffer.

trancode

I/O: Input

STRING:ref:*

Identifies the transaction.

trancode^len

I/O: Input

INT value

Displays the length (in bytes) of the transaction identifier.

ipc^interface

I/O: Input/Output

STRING.EXT:ref:*

Identifies a buffer interface that will act as a place holder for the data^message buffer.

pathsend^write^length

I/O: Output

INT:ref:*

Returns the length (in bytes) of the buffer to be sent to the application server.

error^code

I/O: Output

INT:ref:*

Returns a value showing the state of the call when it completes.

-1

Indicates no errors.

3

Identifies the length of the data that exceeded the maximum buffer size used by the AEF.

error^subcode

I/O: Output

INT:ref:*

Identifies a field always returning a value of -1.

Example

```
!Obtain input data.  In this case input is coming from a terminal device
call writeread^terminal( trm^num
                        ,data^message
                        ,data^length
                        ,diag^message
                        ,diag^length );
!Prepare data buffer to be sent to the application servers
CALL PREPARE^SERVER^IPC( data^message
                        ,data^length
                        ,trancode
                        ,trancode^len
                        ,ipc^interface
                        ,write^length
                        ,error^code
                        ,error^subcode );
! Check for errors
  IF error^code <> -1 then
            call error^routine;
```

Sample Programs

The IT provides the following sample programs to assist you in understanding the use of the IPCO library routines:

- NTCPR1
- NTCPR2

The following sections provide brief descriptions about each program.

NTCPR1

The NTCPR1 sample program is provided as a template to communicate with a CA Gen application using the following logic:

1. Initialize the IPC's information to generate an initial message.
2. Send a message to MSGRT using pathsend.
3. Decode the information returned from MSGRT (basically to retrieve a server class name).
4. Send a message to the server class name. Repeat until there are no changes to another server class.
5. Decode the information from the application (basically data, diagnostic information, or both).
6. Write data or diagnostic information and read your input from the terminal.
7. Prepare a message for the IPC with your input to generate a new message.
8. Repeat Step 2 to Step 7.

The NTCPR1 sample program is built using the following build instructions:

1. Set your current volume to where the IT is located.
2. Issue the following compile command:

   For TNS/R servers

   ```
   pTAL / in ntcpr1s, out $l.#ntc1 / ntcpr1o; symbols, nomap, optimize 0, nowarn 4301,
   fieldalign(platform), srl
   ```

   For TNS/E servers

   ```
   pTAL / in ntcpr1s, out $l.#ntc1 / ntcpr1o; symbols, nomap, optimize 0, nowarn 4301,
   fieldalign(platform)
   ```

3. Issue the following link command:

   For TNS/R servers

   ```
   NLD / name, out $l.#ntcp / ntcpr1o -o ntcpr1 ipco
   ```

   For TNS/E servers

   ```
   ELD / name, out $l.#ntcp / ntcpr1o -o ntcpr1 ipco
   ```

To test with the built NTCPR1, perform the following steps:

1. Set your current volume to the location of your CA Gen application.
2. Start your application by issuing the PWCOLD command.
3. Set your current volume to the location of the NTCPR1 program.
4. Start the NTCPR1 program:

   ```
   NTCPR1
   ```

5. Input the following data:
   - Pathmon name
   - Initial trancode
   - Clear screen options
   - Terminal ID
   - Display subtype (nonExtended or extended)
   - Trace flag
   - User ID
   - Printer name

NTCPR2

The NTCPR2 sample program is provided as a template to communicate with a CA Gen application using the following logic:

1. Initialize the IPC's information to generate a initial message.
2. Send a message to MSGRT using pathsend.
3. Decode the information returned from MSGRT (basically to retrieve a server class name).
4. Send a message to the server class name. Repeat until there are no changes to another server class.
5. Decode the information from the application (basically data, diagnostic information, or both).

6. Write data or diagnostic information and read your input from the terminal.
7. Find the trancode in your input.
8. Prepare a message for the IPC with your input to generate a new message.
9. Repeat Step 4 to Step 8.

The NTCPR2 sample program is built using the following build instructions:

1. Set your current volume to the where the IT is located.
2. Issue the following compile command:

   For TNS/R servers

   ```
   pTAL / in ntcpr2s, out $l.#ntc1 / ntcpr2o; symbols, nomap, optimize 0, nowarn 4301,
   fieldalign(platform), srl
   ```

   For TNS/E servers

   ```
   pTAL / in ntcpr2s, out $l.#ntc2 / ntcpr2o; symbols, nomap, optimize 0, nowarn 4301,
   fieldalign(platform)
   ```
3. Issue the following link command:

   For TNS/R servers

   ```
   NLD / name, out $l.#ntcp / ntcpr2o -o ntcpr2 ipco
   ```

   For TNS/E servers

   ```
   ELD / name, out $l.#ntcp / ntcpr2o -o ntcpr2 ipco
   ```

To test with the built NTCPR2, perform the following steps:

1. Set your current volume to the location of your CA Gen application.
2. Start your application by issuing the PWCOLD command.
3. Set your current volume to the location of the NTCPR2 program.
4. Start the NTCPR2 program:

   ```
   NTCPR2
   ```

5. Input the following data:
   - Pathmon name
   - Initial trancode
   - Clear screen options
   - Terminal ID
   - Display subtype (nonExtended or extended)
   - Trace flag
   - User ID
   - Printer name

Chapter 9: Implementing CA Gen User Exits in NonStop

CA Gen provides support for many system functions so that you do not have to consider them when you design and implement your application. Some of functions are delivered both in source and object form. These functions are called user exits because you can modify the source. Some common uses of user exits are retrieving user IDs, error handling, and site-specific security.

The IT must be installed on your target system before you can modify runtime user exits. The installation process copies the runtime user exits onto your target system and stores them within a source archive named UEXITSC. You can find the source archive in the subvolume where the IT is located.

For NonStop SQL/MX applications, a matching set of user exits are installed in the $IEFH/src directory.

Runtime User Exits

Runtime user exits are standard routines that allow all CA Gen generated applications to access system features. Runtime user exits reside on the target system and CA Gen can access the user exits without actually being coded as part of them. You can use these user exits to supply basic functionality or customize them to the needs of your target system.

In addition to the set of runtime user exits, there is an additional set of distributed processing server user exits. These additional user exits are used only within distributed processing server applications, and are used with distributed processing client user exits that are built into distributed processing client applications. For NonStop SQL/MP, the user exits are supplied in the same source archive UEXITSC. For NonStop SQL/MX, the user exits are supplied in the $IEFH/src directory.

All runtime user exits are provided in both source and object format (in the form of libraries called UEXITCO and DPSUECO) and are loaded onto the target system when the IT is installed. You can use these user exits without changing them because the object format runtime user exits are compiled and are ready to be linked into applications.

Customize User Exits

User exit source code is provided in C so that you can modify the source code to meet your site requirements.

NonStop SQL/MP

The TACL macro MKEXITS is provided to assist with the life cycle of user exits. You can find this macro in the subvolume where the IT is located. You can use MKEXITS to perform the following tasks:

- Extract all user exit source files from the UEXITSC source archive
- Edit a specific user exit source file
- Compile a specific user exit source file
- Rebuild the user exit libraries UEXITCO and DPSUECO

NonStop SQL/MX

The shell scripts $IEFH/make/mkexits and $IEFH/make/mkdpsexit are provided to assist with the life cycle of user exits. You can use these scripts to compile and rebuild the user exit libraries UEXITCO and DPSUECO.

Note: When working with user exits, link all the user exits for a particular library (the set of runtime user exits in UEXITCO or the set of distributed processing server user exits in DPSUECO) regardless of the number of changes as the linker program does not allow you to individually link one exit at a time. Therefore, track the source code changes across all user exits to prevent unwanted functionality to be included in the user exit libraries. If you fail to link all the user exits, unresolved externals show up at execution time.

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

The following user exits are available in C:

Name	Description
TIRYYX	Date User Exit
TIRDCRYP	Decrypt User Exit
TIRDRTL	Default Retry Limit User Exit
TIRNCRYP	Encrypt User Exit
TIRHELP	Help Interface User Exit
TIRUPDB	MBCS Uppercase Translation User Exit

Name	Description
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
TIRSECV	Server Security Validation User Exit
DPSUETDM	Server side user exits for distributed processing server
TIRXINFO	Locale Information User Exit

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

Chapter 10: How to Map a Function Key to the Exit Key Defined for a E107 Application

This appendix explains how to map a function key to the Exit Key defined for an application, and how to map various keyboard keys to specific keys defined for an application as they relate to NonStop.

Exit Key Values

This section lists the Exit Key values that correspond to various function keys you can use on 6530 and 3270 devices. To use a particular function key, you must move the designated graphic character to the EXIT-KEY field within the configuration parameters before shipping that structure to IEFREQ.

For 6530 devices, you can move the designated graphic value to the EXIT-KEY field.

Example:

To use SF16 as the Exit Key, you should code the following statement as part of your initialization of CONFIGURATION-PARAMETERS in the requester you write to call IEFREQ:

```
move "o" to Exit-Key
```

"o"

Defines the graphic character representing SF16

For 3270 devices, you can move the designated graphic value to the EXIT-KEY field of the structure CONFIRMATION-PARAMETERS.

Note: You must confirm that the value moved is the correct ASCII value.

Example:

To use PF24 as the Exit Key, the MOVE statement would be:

```
move "<" to Exit-Key
```

"<"

Defines the graphic character representing PF24.

6530 Function Key Values

The following table lists the function key graphic character representations for 6530 devices.

Function Key	Octal Value	Graphic Character
F1	%100	@
F2	%101	A
F3	%102	B
F4	%103	C
F5	%104	D
F6	%105	E
F7	%106	F
F8	%107	G
F9	%110	H
F10	%111	I
F11	%112	J
F12	%113	K
F13	%114	L
F14	%115	M
F15	%116	N
F16	%117	O
SF1	%140	'
SF2	%141	a
SF3	%142	b
SF4	%143	c
SF5	%144	d
SF6	%145	e
SF7	%146	f
SF8	%147	g
SF9	%150	h
SF10	%151	i

Function Key	Octal Value	Graphic Character
SF11	%152	j
SF12	%153	k
SF13	%154	l
SF14	%155	m
SF15	%156	n
SF16	%157	o

3270 Function Key Values

The following table lists the function key graphic character representations for 3270 devices.

AID EBCDIC (hex)	ASCII (hex)	Graphic Character
PF1 / F1	31	1
PF2 / F2	32	2
PF3 / F3	33	3
PF4 / F4	34	4
PF5 / F5	35	5
PF6 / F6	36	6
PF7 / F7	37	7
PF8 / F8	38	8
PF9 / F9	39	9
PF10 / 7A	3A	:
PF11 / 7B	23	#
PF12 / 7C	40	@
PF13 / C1	41	A
PF14 / C2	42	B
PF15 / C3	43	C
PF16 / C4	44	D
PF17 / C5	45	E
PF18 / C6	46	F

AID EBCDIC (hex)	ASCII (hex)	Graphic Character
PF19 / C7	47	G
PF20 / C8	48	H
PF21 / C9	49	I
PF22 / 4A	5B	symbol for cent
PF23 / 4B	2E	
PF24 / 4C	3C	<
PA1 / 6C	2C	%
CLEAR / 6D	2F	-

How to Map Various Keyboard Keys to Specific Keys Defined for a NonStop Application

The IT lets you map the function keys on a keyboard to the function keys defined for the application through a Keyboard Mapping Table. The Keyboard Mapping Table lets you map 42 possible 6530 function keys to twenty-nine 3270 function keys supported by CA Gen. You can map more than one 6530 function key to the same 3270 function key.

If the 6530 function key is not used by the application, then set its map to 0 or set it to some other 3270 function key that is not used. Unmapped 6530 function keys (map entries that are 0) will simply unlock the keyboard and wait for you to press a mapped 6530 function key.

The Keyboard Mapping Table can use any of the following function keys:

- 16 function key Keyboard Mapping Table
- 12 function key Keyboard Mapping Table

The Keyboard Mapping Table is shipped with the 16 function key, Keyboard Mapping Table enabled.

More information:

How to Use 12 Function Key Keyboard Mapping Table (see page 107)

How to Use 12 Function Key Keyboard Mapping Table

Follow these steps:

1. Remove the comment character (* or --) from lines corresponding to the second set of maps (KEYBOARD-MAP or fkey^map).
2. Add the comment character (* or --) to all lines corresponding to the first set of maps (KEYBOARD-MAP or fkey^map).

Map 6530 Devices Emulating 3270 Devices

There is only one keyboard mapping scheme for 6530 devices emulating 3270 devices. This is shown in the files T3270NS and T3270ES. You can find these files in the subvolume where the IT is located.

For existing applications that use the T3270NS or T3270ES requester, you must edit the call to IEFREQ (in the Procedure Division) to add the parameter KEYBOARD-MAP.

Following is the format of the call command:

```
CALL iefreq
   USING context-session, configuration-parameters, error-link, Keyboard-map
   ON ERROR PERFORM 9999-Exit
```

Map 6530 Devices to Non-TCP Requesters

To map 6530 devices to non-TCP requesters, use the files NTCPR1S or NTCPR2S supplied with the IT as a template for non-TCP requesters. Edit this file by removing the comment character in the section of the Keyboard Mapping Table that is applicable.

Note: Only one keyboard configuration can be supported per requester.

Map 6530 Devices to TCP Requesters

You can use T6530NS or T6530ES files to map 6530 devices to TCP requesters. You can find these files in the subvolume where the IT is located as templates for TCP requesters. You can change the mapping to suit a particular keyboard-application combination.

The following is a sample T6530NS Keyboard Mapping Table, which illustrates standard mapping configurations for a 6530 keyboard.

```
*  6530 Function Key                        Table Offset
*  F1-F16                                  1-16
*  SF1-SF16                                17-32
*  Roll Down (Alt-down-arrow)              33
*  Roll Up   (Alt-up-arrow)                34
*  Page Down (Next Page)                   35
*  Page Up   (Prev Page)                   36
*  Enter                                   37
*  Shift Roll Down (Alt-shift-down-arrow)  38
*  Shift Roll Up   (Alt-shift-up-arrow)    39
*  Shift Page Down (Shift Next Page)       40
*  Shift Page Up   (Shift Prev Page)       41
*  Shift Enter                             42
*
*  3270 Function Key                       Map Value
*  UNMAPPED                                0
*  PF1-PF24                                1-24
*  PA1                                     25
*  PA2                                     26
*  PA3                                     27
*  CLEAR                                   28
*  ENTER                                   29
```

```
*
*  This is the Current IEF Keyboard Map.  There are 42 Entries.
*  THERE MUST be 42 Entries!!!!
*
*  6530 Function Key                       3270 Function Key
*
*  F1-F16                                  PF1-PF16
*  SF1-SF3                                 PA1-PA3
*  SF7-SF14                                PF17-PF24
*  SF15                                    0 (UNMAPPED)
*  SF16                                    0 (UNMAPPED)
*  Roll Down (Alt-down-arrow)              0 (UNMAPPED)
*  Roll Up   (Alt-up-arrow)                0 (UNMAPPED)
*  Page Down (Next Page)                   0 (UNMAPPED)
*  Page Up   (Prev Page)                   CLEAR
*  Enter                                   ENTER
*  Shift Roll Down (Alt-shift-down-arrow)  0 (UNMAPPED)
*  Shift Roll Up   (Alt-shift-up-arrow)    0 (UNMAPPED)
*  Shift Page Down (Shift Next Page)       0 (UNMAPPED)
*  Shift Page Up   (Shift Prev Page)       0 (UNMAPPED)
*  Shift Enter                             0 (UNMAPPED)
*
*
   01 KEYBOARD-MAP.
*       F1 THRU F16 (1-16)
        03 FILLER                       PIC 9(4) COMP VALUE 1.
        03 FILLER                       PIC 9(4) COMP VALUE 2.
        03 FILLER                       PIC 9(4) COMP VALUE 3.
        03 FILLER                       PIC 9(4) COMP VALUE 4.
        03 FILLER                       PIC 9(4) COMP VALUE 5.
        03 FILLER                       PIC 9(4) COMP VALUE 6.
        03 FILLER                       PIC 9(4) COMP VALUE 7.
        03 FILLER                       PIC 9(4) COMP VALUE 8.
        03 FILLER                       PIC 9(4) COMP VALUE 9.
        03 FILLER                       PIC 9(4) COMP VALUE 10.
        03 FILLER                       PIC 9(4) COMP VALUE 11.
        03 FILLER                       PIC 9(4) COMP VALUE 12.
        03 FILLER                       PIC 9(4) COMP VALUE 13.
        03 FILLER                       PIC 9(4) COMP VALUE 14.
        03 FILLER                       PIC 9(4) COMP VALUE 15.
        03 FILLER                       PIC 9(4) COMP VALUE 16.
*       SF1 THRU SF16 (17-32)
        03 FILLER                       PIC 9(4) COMP VALUE 25.
        03 FILLER                       PIC 9(4) COMP VALUE 26.
        03 FILLER                       PIC 9(4) COMP VALUE 27.
        03 FILLER                       PIC 9(4) COMP VALUE 0.
        03 FILLER                       PIC 9(4) COMP VALUE 0.
        03 FILLER                       PIC 9(4) COMP VALUE 0.
        03 FILLER                       PIC 9(4) COMP VALUE 17.
        03 FILLER                       PIC 9(4) COMP VALUE 18.
        03 FILLER                       PIC 9(4) COMP VALUE 19.
        03 FILLER                       PIC 9(4) COMP VALUE 20.
        03 FILLER                       PIC 9(4) COMP VALUE 21.
        03 FILLER                       PIC 9(4) COMP VALUE 22.
        03 FILLER                       PIC 9(4) COMP VALUE 23.
        03 FILLER                       PIC 9(4) COMP VALUE 24.
        03 FILLER                      PIC 9(4) COMP VALUE 0.

        03 FILLER                      PIC 9(4) COMP VALUE 0.

*       ROLL DOWN OR ALT-DOWN ARROW (33)

        03 FILLER                      PIC 9(4) COMP VALUE 0.

*       ROLL UP   OR ALT-UP ARROW (34)

        03 FILLER                      PIC 9(4) COMP VALUE 0.

*       PAGE DOWN OR NEXT-PAGE (35)

        03 FILLER                      PIC 9(4) COMP VALUE 0.

*       PAGE UP   OR PREV-PAGE (36)

        03 FILLER                      PIC 9(4) COMP VALUE 28.

*       ENTER (37)

        03 FILLER                      PIC 9(4) COMP VALUE 29.

*       SHIFT ROLL DOWN OR ALT-SHIFT-DOWN ARROW (38)

        03 FILLER                      PIC 9(4) COMP VALUE 0.

*       SHIFT ROLL UP   OR ALT-SHIFT-UP ARROW (39)

        03 FILLER                      PIC 9(4) COMP VALUE 0.

*       SHIFT PAGE DOWN OR SHIFT NEXT-PAGE (40)

        03 FILLER                      PIC 9(4) COMP VALUE 0.

*       SHIFT PAGE UP   OR SHIFT PREV-PAGE (41)

        03 FILLER                      PIC 9(4) COMP VALUE 0.

*       SHIFT ENTER (42)

        03 FILLER                      PIC 9(4) COMP VALUE 0.

*

*  Alternate Keyboard Mapping (Similar to EM3270)

*

*  To use this Keyboard Mapping Table remove * from lines

*  corresponding to KEYBOARD-MAP below.  Then add an * to all

*  lines corresponding to KEYBOARD-MAP above.
```

```
*
*  6530 Function Key                       3270 Function Key
*
*  F1-F12                                  PF1-PF12
*  F13                                     PA1
*  F14                                     PA2
*  F15                                     PA3
*  F16                                     ENTER
*  SF1-SF12                                PF13-PF24
*  SF13                                    0 (UNMAPPED)
*  SF14                                    CLEAR
*  SF15                                    0 (UNMAPPED)
*  SF16                                    0 (UNMAPPED)
*  Roll Down (Alt-down-arrow)              0 (UNMAPPED)
*  Roll Up   (Alt-up-arrow)                0 (UNMAPPED)
*  Page Down (Next Page)                   0 (UNMAPPED)
*  Page Up   (Prev Page)                   0 (UNMAPPED)
*  Enter                                   0 (UNMAPPED)
*  Shift Roll Down (Alt-shift-down-arrow)  0 (UNMAPPED)
*  Shift Roll Up   (Alt-shift-up-arrow)    0 (UNMAPPED)
*  Shift Page Down (Shift Next Page)       0 (UNMAPPED)
*  Shift Page Up   (Shift Prev Page)       0 (UNMAPPED)
*  Shift Enter                             0 (UNMAPPED)
*
*   01 KEYBOARD-MAP.
**       F1 THRU F16 (1-16)
*        03 FILLER                       PIC 9(4) COMP VALUE 1.
*        03 FILLER                       PIC 9(4) COMP VALUE 2.
*        03 FILLER                        PIC 9(4) COMP VALUE 3.
*        03 FILLER                        PIC 9(4) COMP VALUE 4.
*        03 FILLER                        PIC 9(4) COMP VALUE 5.
*        03 FILLER                        PIC 9(4) COMP VALUE 6.
*        03 FILLER                        PIC 9(4) COMP VALUE 7.
*        03 FILLER                        PIC 9(4) COMP VALUE 8.
*        03 FILLER                        PIC 9(4) COMP VALUE 9.
*        03 FILLER                        PIC 9(4) COMP VALUE 10.
*        03 FILLER                        PIC 9(4) COMP VALUE 11.
*        03 FILLER                        PIC 9(4) COMP VALUE 12.
*        03 FILLER                        PIC 9(4) COMP VALUE 25.
*        03 FILLER                        PIC 9(4) COMP VALUE 26.
*        03 FILLER                        PIC 9(4) COMP VALUE 27.
*        03 FILLER                        PIC 9(4) COMP VALUE 29.
**       SF1 THRU SF16 (17-32)
*        03 FILLER                        PIC 9(4) COMP VALUE 13.
*        03 FILLER                        PIC 9(4) COMP VALUE 14.
*        03 FILLER                        PIC 9(4) COMP VALUE 15.
*        03 FILLER                        PIC 9(4) COMP VALUE 16.
*        03 FILLER                        PIC 9(4) COMP VALUE 17.
*        03 FILLER                        PIC 9(4) COMP VALUE 18.
*        03 FILLER                        PIC 9(4) COMP VALUE 19.
*        03 FILLER                        PIC 9(4) COMP VALUE 20.
*        03 FILLER                        PIC 9(4) COMP VALUE 21.
*        03 FILLER                        PIC 9(4) COMP VALUE 22.
*        03 FILLER                        PIC 9(4) COMP VALUE 23.
*        03 FILLER                        PIC 9(4) COMP VALUE 24.
*        03 FILLER                        PIC 9(4) COMP VALUE 0.
*       03 FILLER                       PIC 9(4) COMP VALUE 28.
*       03 FILLER                       PIC 9(4) COMP VALUE 0.
*       03 FILLER                       PIC 9(4) COMP VALUE 0.
**      ROLL DOWN OR ALT-DOWN ARROW (33)
*       03 FILLER                       PIC 9(4) COMP VALUE 0.
**      ROLL UP   OR ALT-UP ARROW (34)
*       03 FILLER                       PIC 9(4) COMP VALUE 0.
**      PAGE DOWN OR NEXT-PAGE (35)
*       03 FILLER                       PIC 9(4) COMP VALUE 0.
**      PAGE UP   OR PREV-PAGE (36)
*       03 FILLER                       PIC 9(4) COMP VALUE 0.
**      ENTER (37)
*       03 FILLER                       PIC 9(4) COMP VALUE 0.
**      SHIFT ROLL DOWN OR ALT-SHIFT-DOWN ARROW (38)
*       03 FILLER                       PIC 9(4) COMP VALUE 0.
**      SHIFT ROLL UP   OR ALT-SHIFT-UP ARROW (39)
*       03 FILLER                       PIC 9(4) COMP VALUE 0.
**      SHIFT PAGE DOWN OR SHIFT NEXT-PAGE (40)
*       03 FILLER                       PIC 9(4) COMP VALUE 0.
**      SHIFT PAGE UP   OR SHIFT PREV-PAGE (41)
*       03 FILLER                       PIC 9(4) COMP VALUE 0.
**      SHIFT ENTER (42)
*       03 FILLER                       PIC 9(4) COMP VALUE 0.
******************************************************************************
```

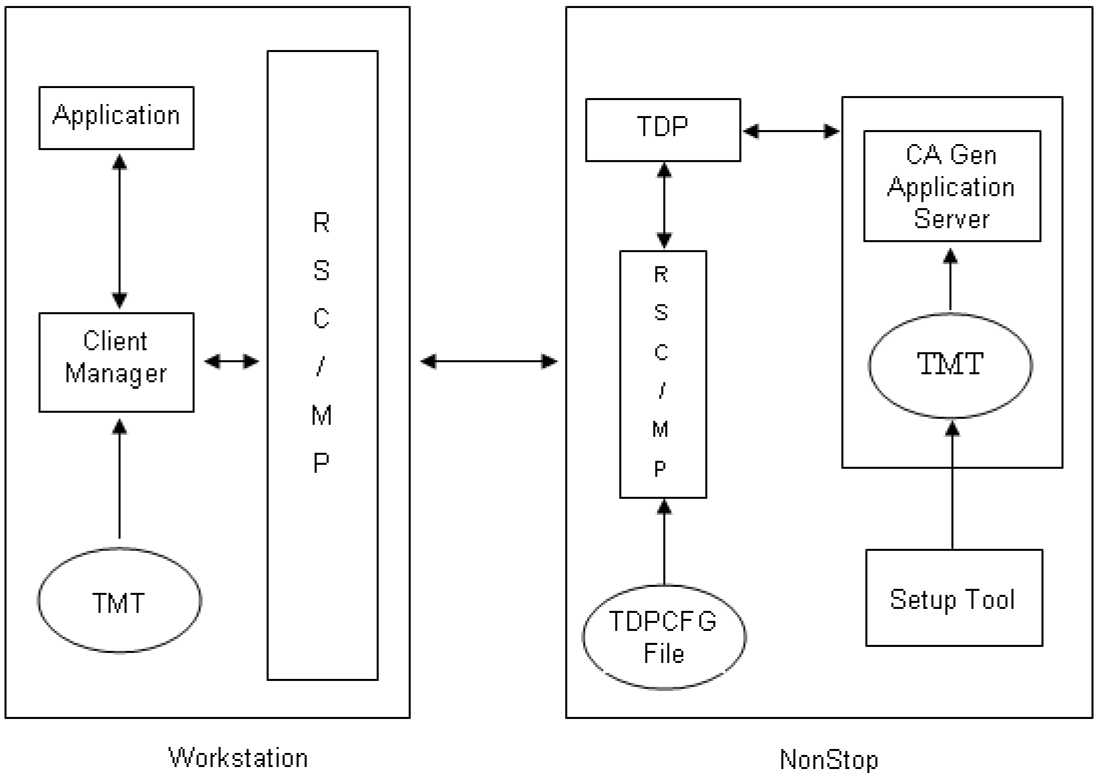
Chapter 11: Configuring Distributed Processing for the NonStop Server

Distributed Processing Environment

The following list details the types of CA Gen application servers built and run on NonStop:

- Block Mode Interactive—These applications use the AEF for execution.
- Block Mode CommandLine NoDisplay—These applications use non-TCP requesters for execution.
- Distributed Processing—These applications communicate with client applications that run on Windows systems.

The following illustration shows the relationship between various components of a distributed processing environment:



To use a distributed processing execution environment, you must complete the following prerequisites:

- Server Configuration
- Workstation Configuration
- Server User Exit Configuration
- Workstation User Exit Configuration

Server Configuration Prerequisites

For distributed processing on NonStop, ensure that the following components are installed, configured, and started:

- Remote Server Call (RSC/MP) over Piccolo
- Transaction Delivery Process (TDP)

Note: For more information about installing, configuring, and starting these products, see the HP *Remote Server Call (RSC/MP) Installation and Management Guide*.

Workstation Configuration Prerequisites

For the Windows workstation to communicate with the NonStop server, ensure that the following components are installed (may use Client Manager or Communications Bridge), configured, and started:

- Remote Server Call (RSC/MP) over Piccolo
- Remove Server Call Initialization file, rsc.ini
- Client Manager to define the NonStop server
- Communications Bridge to define the NonStop server

Note: For more information about installing and configuring RSC/MP over Piccolo, see the HP Remote Server Call (RSC/MP) Installation and Management Guide. For more information about configuring the rsc.ini file and the Client Manager, see the *Distributed Processing—Client Manager User Guide*. For more information about configuring the rsc.ini file and the Communications Bridge, see the *Distributed Processing—Communications Bridge User Guide*.

Server User Exit Configuration

On the NonStop server, a distributed processing user exit (USEREXIT) is provided for encrypting or decrypting transaction data, adding custom data that is not generated by CA Gen to the buffer sent to the client side, customized auditing, and so on.

This user exit is customized, built, and linked into the NonStop application servers that represent the server side of the distributed processing application.

Note: For more information about configuring the server user exit, see the *Distributed Processing—Overview Guide*.

More information:

Implementing CA Gen User Exits in NonStop (see page 99)

Workstation User Exit Configuration

On the Windows workstation, a distributed processing user exit (IORSC.CXX) is provided for encrypting or decrypting transaction data, adding custom data that is not generated by CA Gen to the buffer sent to the server side, customized auditing, and so on.

This user exit is customized, built, and linked into the workstation application that represents the client side of the distributed processing application.

Note: For more information about configuring and building the workstation user exit, see the *Distributed Processing—Overview Guide*.

Chapter 12: Technical Considerations

This appendix provides additional NonStop specific technical information as it relates to CA Gen and the Implementation Toolset.

Handling Data Structures, Error-Link, and RecTime in a NonStop Application

This section describes the data structures used within CA Gen. The following table describes the structures used by program modules and the data structures used in a user-written requester process environment (if it included all the capabilities as supported by the IDS Requester). The last two columns are the EABs, which must be written to extract non-screen information from the generated servers. Some data structures can be eliminated altogether if there are other ways by which certain functions can be handled, such as error logging, statistics, and replay.

CONFIGURATION-PARAMETERS

This structure is only used between the initial requester and IEFREQ so that IEFREQ can perform initialization. If IEFREQ is not being used, you must fill the appropriate structures that use information out of this field.

Sample CONFIGURATION-PARAMETERS Structure:

```
01 CONFIGURATION-PARAMETERS.
02 Display-Terminal          PIC X(16).
02 Display-Terminal-Type     PIC 9.
02 Display-Terminal-Subtype  PIC 9.
02 Debug-Terminal            PIC X(24).
02 Debug-Terminal-Type       PIC 9.
02 Debug-Terminal-Subtype    PIC 9.
02 Initial-TID               PIC X(8).
02 User-Id                   PIC X(16).
02 Commands                  PIC X(78).
02 User-Error-Handler        PIC X(16).
02 Exit-Key                  PIC X.
02 Trace-Flag                PIC 9.
   88 Trace-Flag-ON          VALUE 1.
   88 Trace-Flag-Off         VALUE 0.
02 Test-Flag                 PIC 9.
   88 Test-Flag-ON           VALUE 1.
   88 Test-Flag-Off          VALUE 0.
02 Statistics-Flag           PIC 9.
   88 Statistics-On          VALUE 1.
   88 Statistics-Off         VALUE 0.
02 FILLER                    PIC X.
02 Flag-Capture-Replay       PIC 9.
   88 Flag-Capture-On        VALUE 1.
   88 Flag-Replay-On         VALUE 2.
   88 Flag-Capture-Replay-Off VALUE 0.
02 capture-file              PIC X(8).
02 sequence-number           PIC 9(5) COMP.
02 replay-delay              PIC 9(4) COMP.
02 capture-option            PIC 9.
   88 Flag-Append-On         VALUE 1.
   88 Flag-Append-Off        VALUE 0.
02 Log-Errors-Flag           PIC 9.
   88 Log-All-Errors         VALUE 1.
02 Trace-Level               PIC 9(4) COMP.
02 Idle-Timeout              PIC 9(9) COMP.
02 Max-Retry-Modem           PIC 9(4) COMP.
02 Modem-Delay               PIC 9(4) COMP.
```

DISPLAY-TERMINAL-TYPE

This is a required field for screen devices. Checking the value of DISPLAY-TERMINAL-TYPE determines how a program receives the data—whether to use 6530 or 3270 SEND MESSAGE calls.

The DISPLAY-TERMINAL-TYPE value is sent to several servers and to the IDS-CALLER requester. It is also moved to the field CRTerminal-type of the structure CAPT-TO-SERV.

Values:

- 1—6530 terminal
- 2—3270 terminal

DISPLAY-TERMINAL-SUBTYPE

This is a required field for screen devices. For a true or native 6530 device or one of the 3270 devices that do not support extended attributes, use a value of 1. If the device is a PC running emulation software, or if the 3270 device supports extended attributes, use a value of 2.

The value of DISPLAY-TERMINAL-SUBTYPE is sent to several servers and to the IDS-CALLER requester. This value is also moved to the field CRTerminal-Subtype in the structure CAPT-TO-SERV.

Values:

- 1—Non-extended attributes
- 2—Extended attributes

DEBUG-TERMINAL

This field is obsolete and is no longer used.

DEBUG-TERMINAL-TYPE

This field is obsolete and is no longer used.

DEBUG-TERMINAL-SUBTYPE

This field is obsolete and is no longer used.

INITIAL-TID

The transaction identification value is sent to the MSGRT server to determine the server class name for the initial transaction ID. In typical online applications, this is the transaction identifier for the initial menu screen or the initial logon screen. This value must exist in the TMT table.

USER-ID

This field must receive a unique value for each user. Although this field is 16 characters long, only the last eight non-space characters are used. To prevent multiple operators from appearing to be the same to the system, these eight character positions must be unique. If the field contains spaces, IEFREQ uses the Pathway LOGICAL-TERMINAL-NAME for the AEF as its value.

Clear Screen Options

The commands RESET or RESTART can be sent to the MSGRT server with the initial transaction identifier. The message router currently does not make use of this information.

For use with IDS and 6530 devices, this field is set to spaces. You can send commands such as RESET or RESTART to the runtime. This allows the runtime to restart an application at the point where it ended the last time it was invoked, or to reset the application.

USER-ERROR-HANDLER

This field is obsolete and is no longer used.

EXIT-KEY

This field lets you return control to the initial requester. When IEFREQ identifies the function key that is pressed as the one you specified, it terminates execution and returns control to the initial requester program (the call to IEFREQ completes). There are specialized values available for 6530 and 3270 devices. Move the appropriate value to this field so that IEFREQ can check for that key.

More information:

Exit Key Values (see page 103)

TRACE-FLAG

This value is not used in IEFREQ. It is moved to the field TRACE-FLAG in the CA Gen generated servers. You can use this parameter to determine whether or not the runtime logic in the CA Gen generated server can perform trace. If you set the value to ON, then you need to set an appropriate trace level value in the field TRACE-LEVEL.

Limits: 0 through 255

Values:

- 1—Trace ON
- 0—Trace OFF

TEST-FLAG

This field is obsolete and is no longer used.

STATISTICS-FLAG

If you set this flag, IEFREQ obtains timestamps after transaction receipt, just before sending to the CA Gen generated server, and just after receiving a reply from the CA Gen generated server. This information is formatted and sent to the statistics server, RECSERV. One timestamp is sent to the statistics server for every transaction processed.

Values:

- 1—Statistics ON
- 0—Statistics OFF

FLAG-SWITCH-MESSAGE

This field is moved to the IEFSERV portion of the generated application servers. The FLAG-SWITCH-MESSAGE field determines whether the application servers use flow optimization or not.

Typically, when an application server wants to *dialog flow* to another transaction ID, the application server returns the information back to IEFREQ. The requester issues an END-TRANSACTION verb, starts another transaction with a BEGIN-TRANSACTION verb, and then forwards the transaction to the next appropriate server class. This is how *flow optimization OFF* works.

If you package multiple transaction identifiers into the same server class or if both transactions are contained within the same TMF transaction, you can set flow optimization on. If the new transaction identifier is packaged into the same server class, nothing is returned to IEFREQ. Instead, the server queues up the transaction to the same server. If a new server class is to be used, the flow comes back to the requester, but no new TMF transaction is initiated. To denote flow optimization, use a value of 1.

Values:

- 1—Message ON
- 0—Message OFF

FLAG-CAPTURE-RELAY

You can capture and replay transactions using the following fields:

- FLAG-CAPTURE-RELAY
- CAPTURE-FILE
- SEQUENCE NUMBER
- REPLAY-DELAY
- CAPTURE-OPTION

The field FLAG-CAPTURE-RELAY determines if this feature will be engaged or not. If you do not want to capture or replay, set this field to 0; values in other fields can then be ignored.

To capture transactions, set FLAG-CAPTURE-REPLAY to 1. You must fill in the CAPTURE-FILE field, which tags your capture session with a logical name (this name will be used as the RECAP SQL identifier for a replay session). You also need to place a value in the field CAPTURE-OPTION. If you have already captured some transactions, and you want to add more transactions to the capture file, specify a value of 1. The capture facility then appends additional transactions into the capture session.

To delete any current transactions for a particular user within the capture table, specify a value of 0. This turns append off and causes the Replay server to delete any existing records within the table and starts with a sequence number of 1.

To replay transactions, set FLAG-CAPTURE-REPLAY to 2. You must fill in the CAPTURE-FILE field with the logical name given to an already existing capture session. You must also place a value in the field SEQUENCE-NUMBER to denote the sequence number starts from 1 and increments by 1 from there when captured.

You must specify a value in the REPLAY-DELAY field. This value specifies the number of one-second units to delay before beginning each successive transaction from the capture table. You can ignore the other fields.

Values:

- 0—Replay and capture both OFF
- 1—Capture ON
- 2—Replay ON

CAPTURE-FILE

Use this name to uniquely identify a capture-replay session.

SEQUENCE-NUMBER

This field denotes which capture-replay sequence to use. This occurs only during initialization of the REPLAY mode. This field is not used within the capture mode. Capture appends to the end of the file by determining the current highest-sequence number or by deleting all records for the user and starting again from one.

REPLAY-DELAY

During REPLAY mode, this value is used in the SCOBOL verb DELAY to provide a delay time between the replay of transactions. The value specified in this field is based on one-second units.

CAPTURE-OPTION

This field is used during initialization of the CAPTURE mode. This flag is not otherwise used within the requester. If append ON is designated, additional capture records will be added to the capture file. If append is set to OFF, any records for the current user will be deleted before adding more capture records.

Values:

- 1—Append ON
- 0—Append OFF

LOG-ERRORS-FLAG

Use this field to force all error conditions that IEFREQ is aware of to be written by the LOGSERV server.

Values:

- 0—Do not log errors to LOGSERV but notify LOGSERV with the first and last errors, that is, omit all intermediate errors.
- 1—Notify LOGSERV with all errors.
- All others—Notify LOGSERV with the first and last entry, but not all of the intermediate entries.

TRACE-LEVEL

The numeric value in this field is used to determine the level of tracing carried by the runtime.

Note: Use this field with caution as this will impose a severe performance penalty on the application.

MAX-RETRY-MODEM

This field specifies the maximum number of retries that a SEND MESSAGE operation will be performed in case there is a communication I/O error (error 140) during a read/write operation to a terminal device.

MODEM-DELAY

This field specifies the amount of delay between SEND-MESSAGE operations.

CONTEXT-SESSION

This field is obsolete and is no longer used.

ERROR-LINK

ERROR-LINK is a structure used to pass error information back to the main requester or entry point into the application.

```
01 ERROR-LINK.
     03 GAEF-ERROR                   PIC 9(4) COMP VALUE 0.
     03 T-STATUS                     PIC 9(4) COMP VALUE 0.
     03 T-SUBSTATUS                  PIC 9(5) COMP VALUE 0.
GAEF-ERROR
```

Error messages that are recognized by IEFREQ are placed in GAEF-ERROR and returned to caller requester (the one that called the IDS Requester).

T-STATUS

Displays the Pathway TERMINATION-STATUS value, if any.

T-SUBSTATUS

Displays the Pathway TERMINATION-SUBSTATUS value, if any.

RECTIME

This structure is used to log statistical information. A *write* to the RECSERV server occurs for each transaction processed.

TERMINAL-NAME

Receives the value contained in the Pathway register, LOGICAL-TERMINAL-NAME.

TID

Receives the transaction identifier returned from the message router.

SERVER-NAME

Receives the server class name returned from the message router.

INITIAL-TIME

IEFREQ populates this field with the current time before performing the SEND to the application server call.

ENDING-TIME

IEFREQ populates this field with the current time just after receiving reply from the SEND to the application server.

BYTES-TO-SERVER

The IDS Requester populates this field with the combined data lengths from the fields DATA-LENGTH and DIAGNOSTIC-LENGTH before performing the SEND to the application server.

BYTES-FROM-SERVER

The IDS Requester populates this field with the combined lengths from the fields DATA-LENGTH and DIAGNOSTIC-LENGTH after receiving the reply to the SEND to the application server.

Chapter 13: Using the Applini File

Note: For NonStop SQL/MX applications, the applini file is delivered as application.ini.

Using the Applini File in NonStop

Starting with CA Gen Release 7.6, the runtime utilizes a file named applini. This file holds application-specific environment variables in a token=value pairing. This initialization file is delivered with the C runtime in the CA Gen installation directory. When a C application is built using the Setup Tool or the Build Tool, this file is copied (only once) into the model's executable subvolume/directory. You can customize this copy of the applini file without overriding the copy in the Implementation Toolset location.

When a C application is executed (including distributed processing servers and Block Mode servers), the applini file that resides in the model's executable subvolume/directory is opened and read by the C Runtime. Any uncommented environment variables are then set and used within the application.

Environment Variables in the Applini File

The applini file contains a set of commented environment variables. To use any of these environment variables, you must remove the comment from the variable (";") and modify the value as necessary.

The applini contains the following environment variables:

Environment Variable	Description	Default
TRACE_ENABLE	Enables communications with the Diagram Trace Utility.	1
TRACE_HOST	Defines the Windows system that will run the Diagram Trace Utility.	<none>
TRACE_PORT	Defines the port that the Diagram Trace Utility will listen to.	4567

Note: The Diagram Trace Utility is accessible only if the C application has been generated with Trace Enabled.

Index