COOL:Gen®

Windowed Environments IT User's Guide

6.0

Computer Associates™

Contents

Chapter 1 Implementation Toolset

The *Implementation Toolset* is a collection of tools which resides on the workstation and allows COOL:Gen-generated applications to be installed and run on a target system. The Implementation Toolset (IT) is comprised of the following parts:

- Build Tool
- Application Install
  (required for GUI Runtime application builds)
- Runtime with Diagram Trace
  (GUI or Block Mode, depending on which was purchased for your system)

Build Tool

Use the Build Tool to build, compile, and link COOL:Gen-generated applications from the Construction Toolset development platform in the form of local (*.ICM) or remote (*.RMT) files. You can also use the Build Tool to set up and maintain the test target system. You can invoke it directly or from the Construction Toolset as a local application Build Tool. The Build Tool has a GUI front end but can build COOL:Gen applications targeting GUI or Block Mode screens.

Note: For additional information about invoking the Build Tool directly from the Construction Workstation, see the *COOL:Gen Workstation Construction User Guide*.

Application Install

The Application Install is a set of support files that are required to build GUI-based COOL:Gen-generated applications. These support files include COOL:Gen-specific C header files, libraries, and icons. Block Mode support files are already included in the Block Mode Runtime component.

GUI Runtime with Diagram Trace

The GUI Runtime is a separate, reusable code module and dynamically linked library set that is used to perform low-level functions such as screen formatting and message switching. As the name implies, the GUI (graphical user interface) lets users control the operation of the application through a system of windows, icons, and dialog boxes.

Diagram Trace is used to test the COOL:Gen application by letting you interactively step through action diagrams. This feature lets you view the process and procedure logic used in the diagrams to help in debugging a failed application. Diagram Trace also lets you interactively inspect and modify data views.

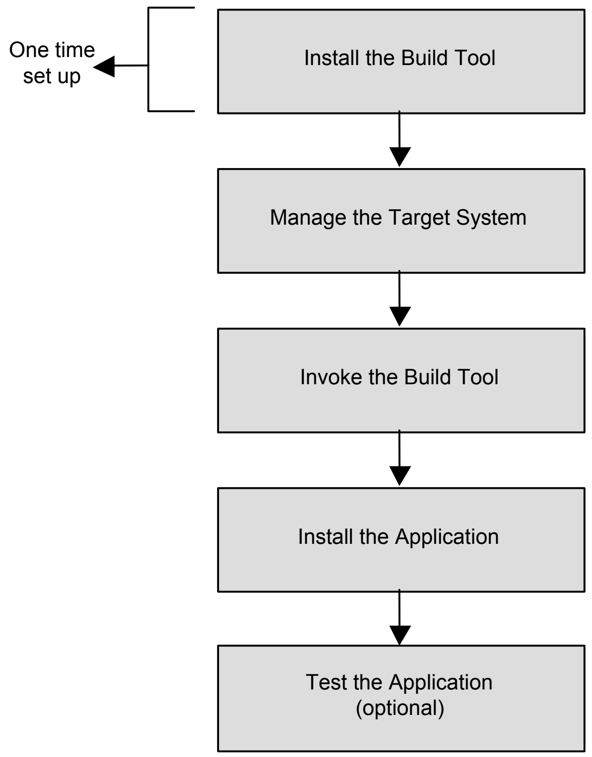
Block Mode Runtime with Diagram Trace

The Block Mode Runtime, like GUI Runtime, is a separate, reusable code package with a dynamically linked library set for Win32 that interfaces between the COOL:Gen-generated application and the operating system of the test target system. Unlike GUI Runtime, you control the application through a set of keystrokes, which are then interpreted by a p3270key mapping file as function definitions. These definitions associate a character string received from the keyboard with a particular 3270 control or function key that determines the character string to be output to the terminal to execute a particular screen function.

Diagram Trace is used to test a COOL:Gen-generated block mode application. This feature lets you view the process and procedure logic used in the diagrams to help in debugging a failed application. If the trace function is being used, then several trace panels display (in addition to the application screens) to let you view the sequence of action block calls. If the trace function is not being used, then the application's screens display exactly as they would appear in a production environment. This feature also lets you step through the execution of an application and view the action diagram statements. And Diagram Trace lets you inspect and modify the data in views.

Required and Optional Tasks

The following illustration shows the required and optional tasks associated with the Build Tool and the order in which they should be performed.



Installing the Build Tool

Before building an application, ensure that you have the Build Tool installed and configured on your target system. For more information, see the *Installing COOL:Gen* guide.

Managing the Target System for Windows NT

During installation, the registry of the target system is modified to provide the proper operating environment for the IT. To verify that the correct system configuration exists for the current logon session, check that:

- This path statement references the COOL\GEN directory. The following illustrates an effective path statement:

  ```
  PATH=...;C:\DEVSTUDIO\COMMON\TOOLS\WINNT;
  C:\DEVSTUDIO\COMMON\MSDEV98\BIN;C\DEVSTUDIO\COMMON\TOOLS;
  C:\DEVSTUDIO\VC98;C:\COOL\GEN;
  ```

- The following user environment variables are properly set:

  ```
  LIB=.;C:\DEVSTUDIO\VC98\MFC\LIB;
  C:\DEVSTUDIO\VC98\LIB;
  C:\COOL\GEN;
  ```

  ```
  INCLUDE=.;C:\DEVSTUDIO\VC98\ALT\INCLUDE;
  C:\DEVSTUDIO\VC98\MFC\INCLUDE;
  D:\DEVSTUDIO\VC98\INCULDE;
  C:\COOL\GEN;
  ```

  ```
  IEF=C:\COOL\GEN\
  ```

  Note: The terminating backslash (\ ) is required.

  ```
  AEHOME=C:\COOL\GEN
  ```

See the Setup section in this guide for information about how to complete the installation.

Specifying Bitmap Locations for GUI Applications

If a GUI application includes bitmaps, the Build Tool searches for a BITMAP subdirectory in the parent of the directory where your models are stored. For example, if your models are in:

```
\MODELS\PROD.IEF\C
```

then the Build Tool searches for bitmaps in:

```
\MODELS\PROD.IEF\BITMAP
```

If the directory does not exist, the Build Tool searches for the files in the current working directory:

```
\MODELS\PROD.IEF\C
```

If you want to specify a directory for bitmaps for all COOL:Gen-generated applications on the system, do **one** of the following:

- To specify a directory for bitmaps which can be changed without rebooting the system, use the Setup button to set the OPT.IEF_BITMAP token to indicate the directory.
- To specify a directory for bitmaps which cannot be changed without rebooting the system (for Win32 only), add the following statement to your AUTOEXEC.BAT file:

  ```
  SET IEF_BITMAP=<bitmap directory>
  ```

The environment variable setting takes precedence over the Setup token IEF_BITMAP.

Building an Application

After the Build Tool has been installed and invoked, you can build your applications. See the “Using the Build Tool” chapter in this guide for more information.

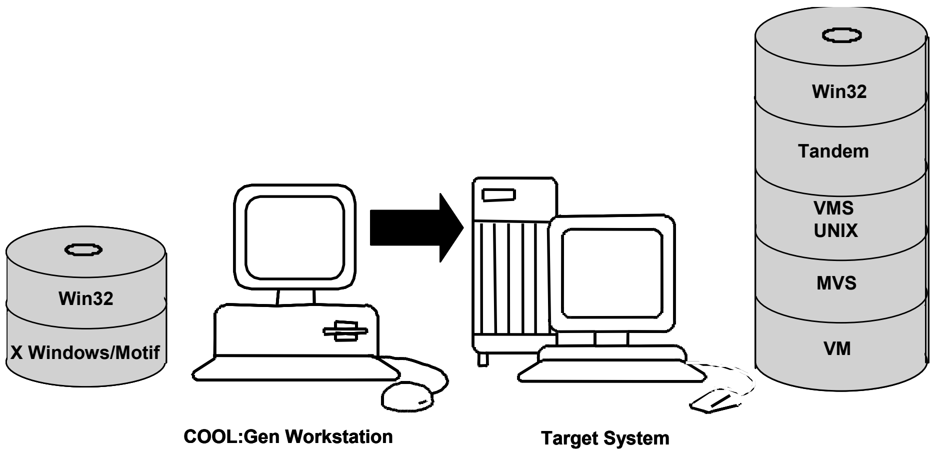
Testing an Application (optional)

Applications that have been built can be moved into production or tested. One method of testing an application is to use the Diagram Trace facility that is provided as part of the Implementation Toolset runtime. See the “Using the Build Tool” chapter in this guide for information required to place an application into production and use the Diagram Trace facility.

Implementing a Target System

To better understand the Build Tool and the principles on which it functions, a short discussion of the concept of target system implementation is provided in the following paragraphs.

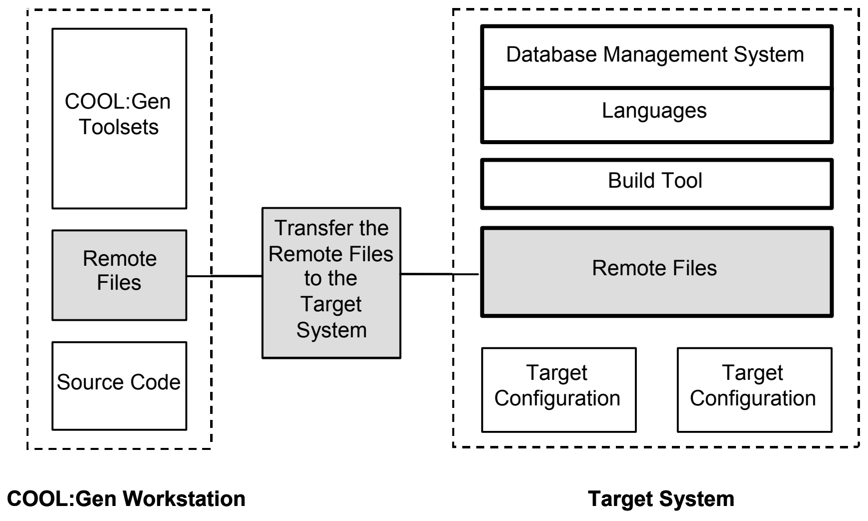
Target system implementation is the process by which an application can be developed on one platform using a particular operating system for subsequent execution on another platform using the same or a different operating system. With this process, a single model can be implemented on many different platforms.



Applications are developed as models on a COOL:Gen workstation using the COOL:Gen toolset. Once constructed, the models are transferred to a target system for execution. During target system implementation, a model is generated on the development platform and processed into special files called local (*.ICM) or remote (*.RMT) files. These files contain source code, data definition language (DDL), and special control information that together allow the model to be installed on the target system within a COOL:Gen environment as a complete executable application. When the Construction Workstation is also the Target System, the component source files are generated along with an Install Control Module (ICM) file for local build. A local installation of an application automatically invokes the Build Tool, which was set up during workstation installation, whereas a remote installation packages the components as remote files for later builds, usually for a remote site.

These components define the organization and contents of the application and make it possible for COOL:Gen to identify and process the application on the target system. The application can then be tested and run on the execution platform. Most applications are comprised of three file types: one for the database information, one for the Referential Integrity (RI) triggers, and one or more executable load modules.

When an application needs to be modified, changes are made to the original model on the development platform. The changes are then regenerated into local or remote files and moved back to the target system. In this manner, enhancements and modifications are made to implemented systems without having to work directly with the generated code.



Target Specifics

So far, we have assumed that to implement a target system, the development platform and execution platform are two separate and distinct machines. However, in Win32 environments, they can also be the same platform. With this arrangement, the step of transferring files from one platform to another is eliminated. This step is also eliminated if shared local area network (LAN) services are used.

Another scenario is the Client/Server Encyclopedia, which can also generate remote files. These remote files must then be built on the target platform into a working application.

Note: See the *COOL:Gen Workstation Construction User Guide* for information about generating and using External Action Block Stubs.

Chapter 2

Using the Build Tool

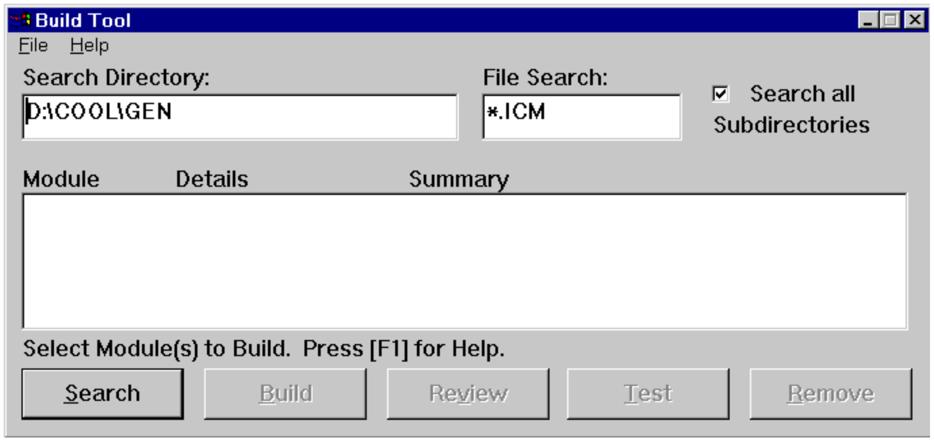
The central component of the Implementation Toolset is the Build Tool. Use it to compile, link, and test COOL:Gen-generated applications.

Build Tool Window

When you invoke the Build Tool, the Build Tool window appears. This menu is used to execute each of the Implementation Toolset tasks.

The following describe each of the window's fields.

Note: A sample remote file is included with the Build Tool to let you test the configuration of the workstation and to let you familiarize yourself with the user interface.



Search Directory: D:\COOL\GEN

File Search: *.ICM

[✓] Search all Subdirectories

Select Module(s) to Build. Press [F1] for Help.

Search Directory Field

The Search Directory field lets you find and identify local or remote files in a specified drive and directory. This field contains the name of the directory to be searched when locating the file types defined by the File Search field.

Although the Search Directory field defaults to the name of the current working directory when the Build Tool is invoked, it can be changed to search any valid path. There is a practical limit of 150 modules, so it is best to reference specific model directories when possible.

Use the following procedures to change the default directory:

Establishing the Default Search Directory for Win32 Systems

1. Open the COOL:Gen icon properties.
2. At the end of the Command Line field of the Program Item Properties window, enter a space and then add the path to the Search directory you want. For example:

   ```
   C:\COOL\GEN\WINIT.EXE C:\COOL\GEN\REMOTE
   ```
3. Click OK.

Two warnings display if you entered the name of an invalid or non-existent directory. The first warning tells you that the directory does not exist and the second tells you to correct the search input. Click OK to close the warning dialog boxes and correct the entry field.

If the default directory entry in the Search Directory field is the directory where the files are located, then you do not have to make changes to this field.

If a directory other than the one shown in the Search Directory field contains the files, then enter the name of the directory into the field.

Search all Subdirectories Check Box

The Search All Subdirectories check box determines whether all the subdirectories of the path defined by the Search Directory field are searched. The default status for this check box is active (checked). Click once on the check box to change its status to inactive (unchecked).

Modules List Box

The Modules list box displays the list of local or remote files and their corresponding modules that have been retrieved as a result of the search.

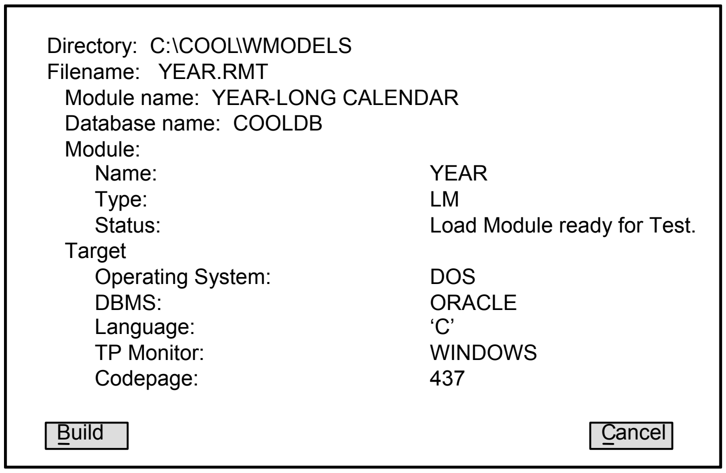
Each non-indented text line in the Modules List Box is the *Model Header* and contains the 32-character model name, the database name, and the directory in which the files reside.

The indented text lines immediately following the model header each represent a separate module that is associated with the parent model. These descriptions begin with the module's 8-character name, followed by its 2-character install control module (ICM) type designation. Next is the module's database management system (DBMS), generator language, and the targeted transaction processing monitor (TP Monitor). The final entry on the line is an informational message describing the status of that particular module. Summaries for some of the modules which could be found in the list are defined in the following table.

Category	Item	Designation	Description
Module			The 8-character name of the module as it appears in the ICM.
Details	ICM Type	DB	The DBMS-specific Database Definition Language (DDL) ICM. This module is not generator language or TP Monitor-specific and is typically processed first.
		RI	This is the Cascade (Referential Integrity) Library ICM. This module type is not TP Monitor-specific but is typically dependent on a DB module. This module is processed after its DB module is installed.
		LM	This is the Load Module (COOL:Gen-packaged executable) with one or more transaction codes. This module contains specific references to all four details and depends on both DB and RI module types.

Category	Item	Designation	Description
	DBMS	IFX	Informix
		ODB	ODBC Database
		ORA	Oracle
		SYB	Sybase
		MSS	Microsoft SQL
	Generator Language	C	C
		COB	Cobol
	TP Monitor	WIN	Windows GUI
		AEF	Block Mode
Summary			The information displayed in this area is used to convey the status of a module to help cue the user for a particular operation. This information changes dynamically, depending on the process being currently performed on the module.

Double-clicking on a single load module description line opens the Module Details window. A sample Load Module Details window is shown following. This window summarizes various information about the selected load module. After inspecting the details text, you can proceed to build the module by clicking Build or you can close the window by clicking Cancel.



```
Directory:  C:\COOL\WMODELS
Filename:   YEAR.RMT
  Module name:  YEAR-LONG CALENDAR
  Database name:  COOLDB
  Module:
     Name:                          YEAR
     Type:                          LM
     Status:                        Load Module ready for Test.
  Target
     Operating System:              DOS
     DBMS:                          ORACLE
     Language:                      'C'
     TP Monitor:                    WINDOWS
     Codepage:                      437
```

Each line in the Modules list box can be individually selected or unselected for building or testing. However, if you single-click on a model header, all of the modules associated with that header are automatically selected (highlighted). Similarly, if you single-click the model header again (while all the modules are highlighted), each of the corresponding load modules are automatically unselected (unhighlighted). If all modules are selected, the Build Tool automatically builds them in the correct order, DB, followed by RI, then LM.

Message Area

Various informational and instructional messages display in this area, depending on the tasks being performed with the Build Tool. The purpose of these messages is to aid you in the execution of these tasks and to direct you to additional options (that is, Online Help).

Build Tool Task Buttons

You use five buttons to execute Build Tool tasks. When you invoke the Build Tool and the Build Tool window appears, each of the buttons appears in an active or inactive (grayed out) state. The state of these buttons changes to reflect their current availability after the validation of an operation, depending on the selections you made in the Modules list box.

Search Button

This button initiates a search for local or remote files to add to the Modules list box. The criteria used to search for local and remote files is determined by the entries made in both the Search Directory and File Search fields. See the Build Tool Window section in this guide for more information about these fields.

A certification of the candidate modules will also prevent modules not intended for the Build Tool's operating system from displaying in the Modules list box.

Build Button

This button builds the modules that are highlighted in the Modules list box; their files are split, their scripts interpreted, and a separate session is launched for each module to complete the build. This button is active only when one or more modules are highlighted in the Modules list box.

Review Button

This button launches the default text editor to display a text file summary of the build for the highlighted module. The results of the compile, precompile, and link steps can be viewed in the scrollable text window. If multiple modules were selected for review, then a separate default text editor session is launched for each module. For Win32, these files can be edited and saved for future reference.

If a module that has not yet been built is highlighted in the Modules list box, then clicking this button returns the message:

```
Module to Review has not been built.
```

Text editor sessions remain open until they are minimized or closed.

This button is active only when one or more load modules are highlighted in the Modules list box.

Test Button

This button opens the Load Module Test window. This window contains a Trancode (Transaction Code) list box, Trace Option selections, and a Clear Screen Input entry field.

This button can only be used when a load module is highlighted in the Modules list box.

Trancode List Box—The Trancode list box contains the module's list of all available trancodes, although only one can be selected at a time.

Trace Option—Selecting Enabled lets you trace the logic of an application while it is being tested. The default state for this option is Disabled. This option is available only if the generator option enabling Trace was set.

Clear Screen Input—This field is used to enter COOL:Gen Clear Screen Input options such as RESTART or RESET.

Remove Button

This button removes the highlighted modules from the list without actually deleting the physical files from the system disk. You can choose to **Remove** files to shorten a long display or to dispense with files that are no longer needed for build, review, or test. This button is activated only when one or more load modules are highlighted in the Modules list box.

To redisplay removed files, define the search parameters for the removed files in the **Searc**h Directory and File Search fields and click Search. The Modules list box will be refreshed and all modules removed during this Build Tool session are redisplayed.

File Menu Options

The File menu contains two options: **Setup** and **Exit**.

Setup Option

Setup opens a default text editor window, from which you can review the contents of the WIN32ITM.TGT (Win32) file to ensure that your system is correctly set up.

Read through the appropriate file for your system and verify that your system reflects what is actually shown in the file.

The following is a sample WIN32ITM.TGT file:

```
; Windows NT/Windows 9X Build Tool - Default target file: WIN32ITM.TGT
; Last Revised: MM/DD/YY ver. X.X
; © Computer Associates International
; This Build Tool setup file provides settings to tailor application
; builds to your specifications in the form of “tokens”.
;
; Any defined “token” record (eg: OPT.DSUSER) can be edited to use
; a customer-supplied “value” (eg: USERID) by editing the value at the end.
; To uncomment a token, delete the ; character preceding a “token” record.
; (Note: For your reference, the first “token” record below is OPT.DSUSER).
;**************[ REQUIRED USER-SPECIFIC IMPLEMENTATION TOKENS ]**************
; ===============( DBMS-dependent DDL/Bindfile Generation )==================
;----------------------------------------------------------------------------
; Environment variables DSUSER and DSPSWD override the following token pair.
; DBMS username and password are configurable per Database.
;
; Oracle Example:
; Replace the values to match a user/password which has been DBA granted
; (eg: DBA commands: GRANT CONNECT,RESOURCE TO user IDENTIFIED BY password;
; GRANT CREATE TABLESPACE TO user;
; GRANT DROP TABLESPACE TO user;
;----------------------------------------------------------------------------
; OPT.DSUSER USERID
; OPT.DSPSWD PASSWORD
;----------------------------------------------------------------------------
; Use the native DBMS utilities to create a server, database, or alias.
; Enter the name in the OPT.DBCONNECT token. If DBCONNECT is not entered
; or is left commented out, the database name stored in the model will
; be used instead.
; Optionally, if DBCONNECT is set to LOCAL, no database name will be used.
; In this case, it is expected that the users local environment is set up
; with a default location to connect to. The method of indicating a
; default location varies by DBMS.
;----------------------------------------------------------------------------
; OPT.DBCONNECT LOCAL
;*********************[ OPTIONAL ADVANCED SETTINGS ]*************************
;****************************************************************************
;****************************************************************************
;**
;** Most setups will not require modification of Advanced Settings.
;** Standard defaults will be overridden when tokens in this category are
;** changed. Caution is advised to prevent any unexpected side-effects.
;**
;** The following list summarizes the Advanced Settings available:
;** - External library: specify a fully-qualified external library filename
;** - Oracle: enable full syntax checking during precompile
;** - MS SQL Server database: create an access module (stored procedure)
;** GUI Application settings
;** - Bitmaps: specify the directory for application *.BMP file resources
;** - rebuild DBMS stub: DBMS version upgrade of libs may require new stub
;** - DLL: build application DLL so action blocks are in a separate DLL
;**
;****************************************************************************
```

```
;---------------------------------------------------------------------------
; The following option specifies the fully qualified external library.
; Remove the comment below to specify the external library if appropriate.
;---------------------------------------------------------------------------
; LOC.EXTERNAL_LIB C:\USER\EXTRNC.LIB
; ===============( GUI Applications Advanced Settings )======================
;---------------------------------------------------------------------------
; An IEF_BITMAP environment variable will provide the bitmap directory.
; If there is no IEF_BITMAP environment variable, the following token,
; OPT.IEF_BITMAP, is used. If OPT.IEF_BITMAP is not enabled (by default),
; then ..\BITMAP subdirectory existence is checked for use as the IEF_BITMAP
; environment variable; otherwise, the module's source directory is used.
;
; Remove the comment from an example below to enable OPT.IEF_BITMAP directory
; Below, in Ex. one, all bitmaps are found in a user-specified directory
; (eg: this may be desirable if bitmaps are shared across network drives).
; In Ex. 2, all local and remote bitmaps are defined to always be processed
; in the source directory (where the module's ICM or RMT file(s) resides.
;---------------------------------------------------------------------------
; OPT.IEF_BITMAP z:\model\win\bitmaps
; OPT.IEF_BITMAP .
;---------------------------------------------------------------------------
; NOTE: GUI applications require DBMS specific startup files (stubs).
; The following begin token definitions for each target DBMS specific
; stub as well as token definitions for how the stub is handled
; during the build process.
; Below is the token definition for building any stub. If OPT.BUILD_SRC is
; set to YES, the build process will copy the DBMS specific stub source to the
; build directory and precompile, compile, and link the stub during the build
; process. Otherwise, the DBMS specific stub executable will be copied into
; the build directory and renamed to {Loadmodule}.EXE. If this option is set
; to yes, the generated window manager resource file is also bound with the
; {Loadmodule}.EXE to provide the executable with the application icons.
; Valid values are [YES | NO].
;---------------------------------------------------------------------------
; OPT.BUILD_SRC NO
;---------------------------------------------------------------------------
; Below is the token definition for build separate Action Block DLLs. If
; OPT.BUILD_AB is set to YES, the build process will separate all action blocks
; from the {LoadModule}.DLL and create a A_{LoadModule}.DLL that is linked
; to the {LoadModule}.DLL. If size of a DLL is at issue, this option can be
; set to YES to reduce the size of the Load Module. Valid values [YES | NO]
;---------------------------------------------------------------------------
; OPT.BUILD_AB NO
;---------------------------------------------------------------------------
; Below is the token definition for building help project files for the
; application's help. If OPT.HELPCOMPILER is set to HCRTF, then the MSDEV 6.x
; help compiler is selected. If OPT.HELPCOMPILER is set to HCW, then MSDEV 4.x
; help compiler is selected. If OPT.HELPCOMPILER is set to HC31 then MSVC 1.5
; help compiler is selected. If the MSVC 1.5 product is not installed copy
; hc31.exe from the MSVC 1.5 cd-rom, directory msvc/bin to a directory in your
; path. The MSDEV 4.x help compiler will fail to compile generated help for
; some GUI applications; the MSVC 1.5 help compiler successfully compiles these
; files. If this token is commented out, then there will be no help compilation.-
; Valid values [HCRTF | HCW | HC31]
;---------------------------------------------------------------------------
; MSVC 6.0
; OPT.HELPCOMPILER HCRTF
; MSVC 4.0
; OPT.HELPCOMPILER HCW
; or
; OPT.HELPCOMPILER HC31
; ===============( DBMS-dependent DDL/Bindfile Generation )==================
;----------------------------------------------------------------------------
; Oracle supports full syntax checking during the precompile cycle. This
; requires DDL to be generated against the DBMS prior to the build cycle and
; connection to the DBMS be available during the build cycle. If full syntax
; checking is desired, the OPT.SQLACCESS token definition should be set to
; YES, the OPT.DBCONNECT token definition above should describe the correct
; connect string, and the OPT.DSUSER and OPT.DSPSWD token definitions above
; should be set to the correct userid/password needs to syntax check against
; the DBMS. If full syntax checking is not desired (Default) then the
; OPT.SQLACCESS token definition should be set to NO.
; Valid values are [YES | NO]
;----------------------------------------------------------------------------
; OPT.SQLACCESS NO
;----------------------------------------------------------------------------
; To create an access module (stored procedure) on a MS SQL Server database,
; uncomment the OPT.SQLACCESS option line above. To specify a non-local
; server, set the OPT.DBCONNECT option line to a server name.
; In addition, to create a bind file, uncomment the OPT.BIND option line.
;----------------------------------------------------------------------------
; OPT.BIND Y
;----------------------------------------------------------------------------
; To force the Build Tool to build GUI applications as if it contains
; embedded SQL even tough the generated code does not, uncomment this token
; and set the value to YES. Valid values are [YES | NO]
;----------------------------------------------------------------------------
; OPT.HAS_SQL NO
;***********[ DEFAULT MIDDLEWARE-DEPENDENT IMPLEMENTATION TOKENS ]***********
;
;-=-=-=-=-=-=-=-=- MQSeries target Middleware -==-=-=-=-=-=-=-=-=-=-=-=-=-=-=
;----------------------------------------------------------------------------
; default LIB path for linking
;----------------------------------------------------------------------------
; LOC.MQSLIB C:\MQM\TOOLS\LIB
;-=-=-=-=-=-=-=-=- DCE target Middleware -==-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
;----------------------------------------------------------------------------
; Default DCE implementation
; Supported values are IBM and GRADIENT
;----------------------------------------------------------------------------
; LOC.DCEPROD GRADIENT
;***************[ DEFAULT DBMS-DEPENDENT IMPLEMENTATION TOKENS ]*************
; ========================( DBMS Settings )==============================
; NOTE: Each DBMS has specific token definitions. If a particular DBMS is
; not used, the token definitions for that DBMS do not need to be set
; or removed. For example, if Oracle is the only target DBMS that will
; be used for generation, there is no reason to set token definitions
; for any other DBMS; removal of these unused tokens is not necessary.
;=-=-=-=-=-=-=-=-=-=-=-=-=- MS SQL Server target DBMS -=-=-=-=-==-=-=-=-=-=-=
;----------------------------------------------------------------------------
; libraries used in the link process
;----------------------------------------------------------------------------
; LOC.NTS95DBLIB SQLAKW32.LIB
; LOC.NTS95SQLLIB CAW32.LIB
;----------------------------------------------------------------------------
; precompiler program
;----------------------------------------------------------------------------
; LOC.NTS95PCC NSQLPREP.EXE
;=-=-=-=-=-=-=-=-=-=-=-=-=-=-=- Oracle target DBMS -=-=-=-=-=-==-=-=-=-=-=-=-
;----------------------------------------------------------------------------
; Oracle version control for WIN95/98
; Oracle 8.1.5 is assumed for COOL:Gen 6.0.
; Oracle 8.0.X is assumed for COOL:Gen 5.1.
```

```
;----------------------------------------------------------------------------
; libraries used in the link process
;----------------------------------------------------------------------------
; LOC.NTORADBLIB ORASQL8.LIB
; LOC.NTORADBLIB SQLLIB80.LIB ; Oracle 8
; LOC.NTORASQLLIB
;----------------------------------------------------------------------------
; precompiler program
;----------------------------------------------------------------------------
; LOC.NTORAPCC PROC.EXE
; LOC.NTORAPCC PROC80.EXE ; Oracle 8
;----------------------------------------------------------------------------
; default INCLUDE path for compiling
;----------------------------------------------------------------------------
; LOC.NTORAINCLUDE C:\ORACLE\ORA81\PRECOMP\PUBLIC
; LOC.NTORAINCLUDE C:\ORANT\PRO80\C ; Oracle 8
;----------------------------------------------------------------------------
; default LIB path for linking
;----------------------------------------------------------------------------
; LOC.NTORALIB C:\ORACLE\ORA81\PRECOMP\LIB\MSVC
; LOC.NTORALIB C:\ORANT\PRO80\LIB\MSVC ; Oracle 8
;----------------------------------------------------------------------------
; IRECLEN and ORECLEN options for precompiling
;----------------------------------------------------------------------------
; OPT.IRECLEN 255
; OPT.ORECLEN 255
;=-=-=-=-=-=-=-=-=-=-=-=-=-=-=- XDB target DBMS -=-=-=-=-=-==-=-=-=-=-=-=-=-=
;----------------------------------------------------------------------------
; libraries used in the link process
;----------------------------------------------------------------------------
; LOC.NTXDBDBLIB PREC32N.LIB
; LOC.NTXDBSQLLIB
;----------------------------------------------------------------------------
; precompiler name
;----------------------------------------------------------------------------
; LOC.NTXDBPCC PRECOMP.EXE
;
;=-=-=-=-=-=-=-=-=-=-=-=-=- INFORMIX target DBMS -=-=-=-=-=-==-=-=-=-=-=-=-=-
;----------------------------------------------------------------------------
; libraries used in the link process
;----------------------------------------------------------------------------
; LOC.NTINFDBLIB ISQLT09A.LIB
; LOC.NTINFSQLLIB
;----------------------------------------------------------------------------
; precompiler name
;----------------------------------------------------------------------------
; LOC.NTINFPCC ESQL.EXE
;=-=-=-=-=-=-=-=-=-=-=-=- SYBASE target DBMS -=-=-=-==-=-=-=-=-=-=-=-=-=-=-=-
;----------------------------------------------------------------------------
; libraries used in the link process
;----------------------------------------------------------------------------
; LOC.NTSYBDBLIB LIBCT.LIB
; LOC.NTSYBSQLLIB LIBCS.LIB
;----------------------------------------------------------------------------
; precompiler program
;----------------------------------------------------------------------------
; LOC.NTSYBPCC CPRE.EXE
;----------------------------------------------------------------------------
; default INCLUDE path for compiling
;----------------------------------------------------------------------------
; LOC.NTSYBINCLUDE C:\SYBASE\OCS-12_0\INCLUDE
;----------------------------------------------------------------------------
; default LIB path for linking
;----------------------------------------------------------------------------
; LOC.NTSYBLIB C:\SYBASE\OCS0-12_0\LIB
```

```
;=-=-=-=-=-=-=-=-=-=-=-=- DB2/2 target DBMS -=-=-=-==--=--=-=-=-=-=-=-=-=-=-=
;-----------------------------------------------------------------------------
; libraries used in the link process
;-----------------------------------------------------------------------------
; LOC.NTDB2DBLIB DB2API.LIB
; LOC.NTDB2SQLLIB
;-----------------------------------------------------------------------------
; precompiler program
;-----------------------------------------------------------------------------
; LOC.NTDB2PCC TIDB2PRP.EXE
;-----------------------------------------------------------------------------
; default INCLUDE path for compiling
;-----------------------------------------------------------------------------
; LOC.NTDB2INCLUDE C:\SQLLIB\INCLUDE
;-----------------------------------------------------------------------------
; default LIB path for linking
;-----------------------------------------------------------------------------
; LOC.NTDB2LIB C:\SQLLIB\LIB
;=-=-=-=-=-=-=-=-=-=-=-=- ODBC target DBMS -=-=-=-==--=--=-=-=-=-=-=-=-=-=-=-
;-----------------------------------------------------------------------------
; libraries used in the link process
;-----------------------------------------------------------------------------
; LOC.NTODBCDBLIB ODBC32.LIB
; LOC.NTODBCSQLLIB TIODBC.LIB
;-----------------------------------------------------------------------------
; default INCLUDE path for compiling
;-----------------------------------------------------------------------------
; LOC.NTODBCINCLUDE
;-----------------------------------------------------------------------------
; default LIB path for linking
;-----------------------------------------------------------------------------
; LOC.NTODBCLIB
;-----------------------------------------------------------------------------
; Block mode load modules running in aefad/aefcn
; To build a block mode load module in a format which
; allows it to be invoked via aefad/aefcn, uncomment
; the OPT.TE token set to AEFAD.
;-----------------------------------------------------------------------------
; OPT.TE AEFAD
;*****************************[ CBD OPTIONS ]*********************************
;-----------------------------------------------------------------------------
; For component consumption, uncomment the following
; token and set it to the name of a text file
; which lists component LIBS to use at link time.
;-----------------------------------------------------------------------------
; OPT.CBDLIST
;*****************************[ PROXY OPTIONS ]*******************************
;=-=-=-=-=-=-=-=-=-=-=-=- Java Proxy -=-=-=-==--=--=-=-=-=-=-=-=-=-=-=-=-=-=-
;-----------------------------------------------------------------------------
; Normally, the generated jar files for the Java proxy contain
; the COOL:Gen runtime within them. To have the jar files
; built with only the generated code, uncomment the following
; token and set its value to NO. Valid values are [YES | NO]
;-----------------------------------------------------------------------------
; OPT.JARRUNTIME YES
;-----------------------------------------------------------------------------
; By default, COOL:Gen will build the appropriate jar files
; for the Abean and Applet/Servlet combinations. To only build
; a specific one, uncomment out the undesired one and set its
; value to NO. Valid values are [YES | NO]
```

```
;--------------------------------------------------------------------------
; OPT.BUILDABEAN YES
; OPT.BUILDSERVLET YES
;--------------------------------------------------------------------------
; When building the Applet/Servlet combination of the Java
; proxy, the location of the javax.servlet.* classes must be
; specified. The actual classes may be provided from a variety
; 3rd party vendors. You may specify a directory or a jar file
; where the classes are located. Uncomment the following
; token and set its value to the location of the class files.
;--------------------------------------------------------------------------
; LOC.SERVLETCLASSES
;--------------------------------------------------------------------------
; When building an eBusiness application, you can set the following
; LOC tokens to indicate the location where you want
; particular files to be copied when a successful build has
; occurred.
;--------------------------------------------------------------------------
; Local web server directory to which the DLL will be copied.
; This directory is from the perspective of the web server machine.
; LOC.LOCAL_DLLDIR
;
; Web server directory to which the DLL will be copied.
; This directory is from the perspective of the build machine.
; LOC.DLLDIR
;
; Web server directory to which the contents of the html directory will be
; copied.
; This directory is from the perspective of the build machine.
; LOC.WEBSRVRDOC
;
; Web server directory to which the generated .class files will be copied.
; This directory is from the perspective of the build machine.
; LOC.SERVLETDIR
;
; END OF WIN32ITM.TGT
```

Exit Option

This option is used to exit the Build Tool.

If you select Exit while a module is being built, the build is allowed to complete, though the Build Tool is immediately exited.

Processing Local or Remote Files

To create an executable application from the local or remote files transferred from the development platform, use the following procedure:

1. Invoke the Build Tool.
2. If necessary, change the Search path and type of file to find (.ICM or .RMT) then click the Search button to display the modules you want to process.

3. Select the Model Header in the Modules List Box that represents the application you want to build.

   All the load modules corresponding to the selected model are highlighted.

   Note: If there are any graphics files that accompany the application, move them with the .RMT files.

4. Click Build. The modules are built sequentially.

   A session with the title "Application Install" is automatically launched to provide a dynamic display of the build process. Check the session display for possible errors before closing the session.

   The Summary section of the load module description in the Modules List Box changes to Review results when Build is done.

5. Click Review after the build process is completed.

   A separate text editor session is launched that corresponds to each module that has been built. These windows contain text files that represent the detailed results of the various build steps (compile, precompile, and link) that have been performed on the load modules. Review each file to discern the success or failure of a particular build.

   If any errors are detected during build, they are described in these files. Error statements are preceded by multiple asterisks (***).

6. Close each text editor or Browse session after reviewing it for possible errors.

7. Test the application.

Testing an Application

To test a successfully built application, use the following procedure.

1. Unselect all highlighted modules.
2. Highlight a single load module to be tested.
3. Click Test. The Load Module Test window is displayed.
4. Enter any Clear Screen inputs required for your application.
5. Highlight a single trancode and click OK.

Appendix

A Customizing Scripts

Scripts are tokenized command procedures that contain all the specific commands necessary to perform the compile and link steps for a module.

Note: Editing scripts is not generally recommended. COOL:Gen-provided scripts are tested for supported targets and should be sufficient for general use. However, we recognize that specific user needs can vary and newer target versions might require that existing scripts be modified. In this case, any changes to a script should be made only by a system administrator who is familiar with any possible downstream effects which might occur as a result of making the changes.

During the build of a module, the Build Tool interprets the script and replaces the tokens with information about the local or remote file from the setup information and produces a complete command procedure. The Build Tool then submits the command procedure for execution.

Tokens are generic placeholders that are used for substitution in a script. When a script is being interpreted, each token is resolved to its representative value so that the resulting command procedure is specific to a module and its set-up environment.

The information that appears as tokens in a script include:

- Information about the elements of the module itself. This information is taken from the Install Control Module (ICM) portion of the file.
- Locations and target configuration information defined previously using the Build Tool.
- Environment (ENV) and Option (OPT) tokens. Environment tokens come from the script definition. Option tokens are user-defined in the WIN32ITM.TGT (Windows NT) or in the build startup command file WIN32BLD.BAT (Windows NT).

When the script is processed, unresolved tokens are resolved as a NULL value.

Note: Structures of tokens within tokens are **not** supported.

Locating Scripts

The Build Tool contains a set of valid basic scripts for a specific operating environment. You can load them as they are or customize them for a particular application.

Script Naming Convention

COOL:Gen script files are specific to the target system hardware, operating system, DBMS, and computer language being used. The hardware, operating system, DBMS, language, and type of each script file is coded into the name of the scripts according to the following convention:

Example of Script Naming

Consider the following as a sample script name: *intdbltt*

intdb—Identifies scripts for Windows NT environments.

l—Identifies the language. The following languages are available:

- c—C language
- j—Java language

tt—Identifies the type of script. The following script types are available:

- jp—Java Proxy
- db—Database DDL
- lm—Load Module
- ri—Referential Integrity Triggers
- cp—Com Proxy

Script File Name (.SCR)	Language	Type
intdbccp	C	Com Proxy
intdbcdb	C	Database DDL
intdbcjp	C	Java Proxy
intdbclm	C	Load Module
intdbcri	C	RI Triggers
intdbjlm	Java	Load Module
intdbjri	Java	RI Triggers

Token Categories

There are three categories of information that can appear as tokens in a script. These are shown in the following table.

Category	Type of Information	Prefix
Environment Parameter Tokens	This is information that defines the specific compile and build options for a specific Target Configuration.	ENV or OPT
GML Tokens	This is information about elements of the file itself. This is taken from the ICM portion of the file.	LMD or LMM
Location Tokens	These are the directory locations that have been defined previously using the Build Tool, and other configuration definition information (database name, etc.).	BUS MOD, LIB TAR, or LOC

The following list describes the contents of each token prefix type, with token category descriptions being presented in the sections following the table.

BUS—These are BusSys tokens. The values for these tokens are set in the ICM.

ENV—These are environment definition tokens. These tokens describe characteristics of a specific Target Configuration. They are created from information shown in the WIN32ITM.TGT file.

LMD—These are load module definition tokens. These tokens describe characteristics of the contents of a load module. They come from the ICM of a file.

LMM—These are load module member tokens. These tokens describe characteristics of one portion of a load module (such as a procedure step or an action block). They come from the ICM of a file.

LOC—These are location tokens. The values for these tokens are user-defined in the WIN32ITM.TGT file, or in the build startup command WIN32BLD.BAT.

MOD—These are model tokens. The values are set in the ICM.

OPT—These are option tokens. The values are user-defined in the WIN32ITM.TGT file, or in the build startup command file, WIN32BLD.BAT.

TAR—These are Target Configuration tokens. The values for these tokens are set in the ICM.

When the script is processed, unresolved tokens are resolved as a NULL value. Structures of tokens within tokens are not supported.

Environment Parameter Tokens

The two categories of environment parameter tokens are:

- Environment Tokens
- Option Tokens

Environment Tokens

Environment tokens identify the variable components of a Target Configuration definition. These elements include the DBMS, code generation language, teleprocessing monitor, and screen format used in one Target Configuration. Environment tokens are identified by an **ENV** prefix. Environment tokens are not used in the default scripts provided for the PC environment.

Option Tokens

Option tokens are user-defined tokens that can be used to define additional types of locations, specify options when invoking compilers or other programs, or provide other capabilities. Option tokens are identified by the **OPT** prefix. Option tokens can be assigned specific values for a Target Configuration.

GML Tokens

Generalized Markup Language (GML) is the language used in the ICM of each COOL:Gen-generated local or remote file. The information in the ICM defines the contents of the database, RI trigger set, or load module local or remote file. For database and RI trigger local or remote files, all of the components specified in the ICM are found in the file. For a load module local or remote file, the complete load module is defined in the ICM, regardless of the number of components included in the file. (This is done for maintenance purposes. It lets you change and regenerate portions of a load module without changing its definition.)

Note: An incomplete load module ICM can result if the subset used to generate the file does not contain all the elements for the load module.

Tokens in GML are represented as equate statements, which have an identifier to the left of the equal sign and the variable data to the right of the equal sign. Not all of the equate statements shown in an ICM are used as tokens by the script.

When the equate statements are processed into tokens, they are assigned to one of the GML token types. The GML token types generated by COOL:Gen are:

- BUS
- LMD
- LMM
- MOD

Location Tokens

When a build command procedure is created as a part of a local or remote file build, the location tokens are replaced with the actual locations where COOL:Gen-generated components should be stored when the file is built.

Location tokens are typically identified by the prefix, LOC.

The default hierarchy logic in the Build Tool searches for a location at the lowest (most specific) level first, then moves up one level at a time to more general levels until it finds a location specification. The following is the search order for the directory hierarchy:

1. Business System level specific category locations
2. Business System level default location
3. Model-level specific category locations
4. Model-level default location
5. Target-level specific category locations
6. Target-level default location.

This search order is transparent when you are viewing a script. It might be necessary to use a specific location rather than resolving a token through the location hierarchy. Special tokens are available to reference the default locations, external action block libraries, and code libraries. These tokens are provided in addition to the **LOC** tokens. There names are:

- BUS.DEFAULT
- BUS.EXTERNAL_LIB
- MOD.DEFAULT
- MOD.EXTERNAL_LIB

- TAR.DEFAULT
- TAR.EXTERNAL_LIB

All tokens are listed in the following table.

Type	Name	Description
BUS	DEFAULT	The default location that contains all components for which no other location is specified.
BUS	EXTERNAL_LIB	The name of the external action block library.
BUS	SOURCE	The source directory on the development workstation. This value is found in the :systems section of the ICM of each file associated with this BusSys as source=.
BUS	TECHSYS	The name of the BusSys. This value is found in the :systems section of the ICM of each file associated with this BusSys as techsys=.
LMD	DATE	The date when the code was generated. This value is found in the :execdef section of the ICM of each file as date=.
LMD	DATEUP	The date when the Model was uploaded to the COOL:Gen central encyclopedia. This value is found in the :source section of the ICM of each file as dateup=.
LMD	DBMS	This is the DBMS this file was created for. This value is found in the :execunit section of the ICM of each file as dbms=.
LMD	DBNAME	The name of the database associated with this file. This value is assigned during code generation and is found in the :execunit section of the ICM of each file as dbname=.
LMD	EXECENV	This is the execution environment (teleprocessing monitor) for this file. This value is found in the :execunit section of the ICM of each load module file as execenv=.
LMD	FORMAT	The screen format for this file. This value is found in the :execunit section of the ICM of each load module file as format=. If the screen format is an IBM 3270 emulator (as for the AEF), the value in this field is BYPASS. If the screen format is a forms package native to the target system, the value in this field is MAPPED.
LMD	GEN_OS	The operating system where the file was generated. The value for this token is found in the :execdef section of the ICM of each file as os=.
LMD	LANGUAGE	The compiler language used for this file. This value is found in the :execunit section of the ICM of each file as language=.

Type	Name	Description
LMD	LEVEL	The levelset of the COOL:Gen Code Generation Tool used to create the file. This value is found in the :source section of the ICM of each file as level=.
LMD	MEMBER	The member name (load module name). This value is found in the :execunit section of the ICM of each load module file as member=.
LMD	MODEL	The model used to create this file. This value is found in the :source section of the ICM of each file as model=.
LMD	OS	The operating system for which the file was created. This value is found in the :execunit section of the ICM of each file as os=.
LMD	PROFILE	The user's profile type. This value is found in the :execunit section of the ICM of each file as profile=.
LMD	SAVED	This value is found in the :source section of the ICM of each file as saved=.
LMD	SCHEMA	The COOL:Gen schema level. This value is found in the :source section of the ICM of each file as schema=.
LMD	SOURCE	The source directory for the components used to create the file on the development workstation. This value is found in the :systems section of the ICM of each file as source=.
LMD	SUBSET	The model subset used to create this file. This value is found in the :source section of the ICM of each file as subset=.
LMD	USER	The user who created the file. The value for this token is found in the :execdef section of the ICM of each file as user=.
LMD	TECHSYS	The BusSys that owns the load module. This value is found in the first :pstep section of the ICM of each load module file as techsys=.
LMD	TIME	The time when the file was created. The value for this token is found in the :execdef section of the ICM of each file as time=.
LMD	TIMEUP	The time when the model was uploaded to the COOL:Gen central encyclopedia. This value is found in the :source section of the ICM of each file as timeup=.
LMM	ABTYPE	The action block type. Appears only if the load module member is an action block.
LMM	CLRTRAN	The clear screen transaction code. Appears only if the load module member is a procedure step. The value for this token is found in the :pstep definitions in the ICM as clrtran=.

Type	Name	Description
LMM	CREATE	Indicates whether this member can perform an SQL CREATE statement. The value for this token is found in the :pstep, :acblk, :entity, and :relation definitions in the ICM of load module and RI files as create=.
LMM	DELETE	Indicates whether this member can perform an SQL DELETE statement. The value for this token is found in the :pstep, :acblk, :entity, and :relation definitions in the ICM of load module and RI files as delete=.
LMM	DLGTRAN	The dialog flow transaction code. Appears only if the load module member is a procedure step. The value for this token is found in the :pstep definitions in the ICM as dlgtran=.
LMM	MEMBER	The member name (file name) for this procedure, screen, or action block. The value for this token is found in the :pstep, :screen, :acblk, :entity, and :relation definitions in the ICM of load module and RI files as member=.
LMM	NAME	The name of the procedure, screen, or action block. The value for this token is found in the :pstep, :screen, and :acblk definitions in the ICM as name=.
LMM	SQL	Indicates whether this member has embedded SQL. The value for this token is found in the :pstep, :acblk, :entity, and :relation definitions in the ICM of load module and RI files as sql=.
LMM	TECHSYS	The BusSys where this load module member belongs. The value for this token is found in the :pstep definitions, :screen definitions, and :acblk definitions in the ICM as techsys=.
LMM	TEST	Indicates whether this procedure step or action block was created with the diagram test capabilities. The value for this token is found in the :pstep and :acblk definitions in the ICM as test=.
LMM	TYPE	Indicates the load module member's type. The value for this token is determined by the name of the section in the ICM. Examples of types are :pstep, :screen, and :acblk.
LMM	UPDATE	Indicates whether this member can perform an SQL UPDATE statement. The value for this token is found in the :pstep, :acblk, :entity, and :relation definitions in the ICM of load module and RI files as update=.
LOC	CODE_EXE	Identifies the location where the load modules were originally created. This token is used only during the build process and is not intended for use during test and execution.

Type	Name	Description
LOC	CODE_LIB	Identifies the location for object code for individual components in library form. This token can occur at the Target Configuration, Model, and BusSys levels.
LOC	CODE_LIS	Identifies the location for listings produced during the compile and link process. This token can occur at the Target Configuration, Model, and BusSys levels.
LOC	CODE_OBJ	Identifies the location for object code for individual components. This token can occur at the Target Configuration, Model, and BusSys levels.
LOC	CODE_SRC	Identifies the location for source code for individual components. This token can occur at the Target Configuration, Model, and BusSys levels.
LOC	DDL	Identifies the location for the DDL used to create the database. This token can occur at the Target Configuration and Model levels.
LOC	DEFAULT	Identifies the default location that contains all components for which no other location is specified. This mandatory token occurs at the Target Configuration, Model, and BusSys levels.
LOC	ICM	Identifies the location for the ICM files for installed files. This token can occur at the Target Configuration, Model, and BusSys levels.
LOC	MAKE_OUTPUT	Identifies the location for the output file produced when the command procedure is run for a file. This token can occur at the Target Configuration, Model, and BusSys levels.
LOC	MAKE_PROC	Identifies the location for the command procedures produced by interpreting the script for each file that is installed. This token can occur at the Target Configuration, Model, and BusSys levels.
LOC	MAKE_STATUS	Identifies the location for the status file produced when the command procedure is run for a file. This token can occur at the Target Configuration, Model, and BusSys levels.
LOC	RI_TRIG_LIB	Identifies the location for the RI trigger routines in library form. This token can occur at the Target Configuration and Model levels.
LOC	RI_TRIG_LIS	Identifies the location for compiler listings produced when installing RI trigger routines. This token can occur at the Target Configuration and Model levels.
LOC	RI_TRIG_OBJ	Identifies the location for RI trigger routines in object form. This token can occur at the Target Configuration and Model levels.
LOC	RI_TRIG_SRC	Identifies the location for RI trigger routines in source form. This token can occur at the Target Configuration and Model levels.

Type	Name	Description
MOD	DBNAME	For Oracle, this token contains the authorization ID used to create the database. For Ingres, this is the name of the database to create.
MOD	DEFAULT	Identifies the default location that contains all components for which no other location is specified.
MOD	EXTERNAL_LIB	The fully-qualified name of the archive library of external action blocks.
TAR	DBNAME	For Oracle, this token contains the authorization ID used to create the database. For Ingres, this is the name of the database to create.
TAR	DEFAULT	Identifies the default location that contains all components for which no other location is specified. At the Target Configuration level, this location token is mandatory.
TAR	EXTERNAL_LIB	The fully-qualified name of the archive library of external action blocks.

Token Delimiters

Token delimiters are detailed in the following table. Note that an additional category of tokens can be added to scripts to provide additional capabilities not available using standard tokens.

Description	Start	End
Script Token Delimiter	{	}
Script Comment Delimiter	{*	*}
Script Directive Delimiter	{[	
Script Carriage Return Suppression	{-}	]}

Customizing Scripts

COOL:Gen scripts can be modified using any standard ascii text editor to fulfill the requirements of a particular target system. Modifications can include changes to directory locations, such as those used by the DBMS precompiler and the addition of new option tokens.

Note: Editing scripts is not generally recommended. COOL:Gen-provided scripts are tested for supported targets and should be sufficient for general use. However, specific user needs can vary or newer target versions can require that existing scripts be modified. In this case, any changes to a script should only be made by a system administrator who is familiar with any possible downstream effects that might occur as a result of making the changes.

Script Delimiters

You can specify the delimiters to be used for each type of delimited string that appears in a script. This must be done before the first OPENFILE script directive. Delimiters are specified when a script is loaded and are shown on the screen when the script displays. The default values for the token delimiters are shown in the Default Script Delimiters table. Use the extra space in the table to record the delimiters for your target system if they differ from those shown by the table.

Script Directives

A number of different script directives are used to control the activity within a script. Some allow use of a looping structure to perform an action on every occurrence of a specified type while others let you create a module token or control the flow of logic from a compare operation. The following script directives are supported in this release:

- DEFINE
- ELSE_IF
- ENDIF
- FOREACH...ENDFOR
- IF_EITHER_EXISTS
- IF_EQUAL
- IF_EXISTS
- IF_NOT_EQUAL
- INCLUDE
- OPENFILE...CLOSEFILE

Note: It is important to make sure that the script will still work properly if the option token values are not found and tokens are resolved to nulls.

Appendix B Working with User Exits

Note: The Implementation Toolset must be installed on your target system before User Exits can be modified. The installation process copies the User Exits onto your target system and stores them in the COOL\GEN directory. For information about installation, see the *Installing COOL:Gen* guide. Modifying User Exits is an optional task.

COOL:Gen supports certain system functions, such as retrieving a user ID, which can vary in implementation from target system to target system because of the combination of hardware and software being used in the configuration of that target system.

User exits are standard routines which allow all COOL:Gen-generated applications to access these system features. User exit routines reside on the target system and can thus be accessed by each COOL:Gen 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.

All User Exits are provided in both source and object format and loaded onto the target system when the Implementation Toolset is installed. The object-format User Exits are ready to be linked into applications as they are compiled and can be used without change. The source code is provided in all supported languages, depending on your target environment, so you can modify User Exits to meet your site's requirements. When the modifications are completed, compile the modified exit, make a backup copy of the existing exit library, then overwrite the existing version with your new one. All applications that use the recompiled exit will have to be relinked before the change becomes effective.

There are two sets of User Exits: those for GUI applications and those for Block Mode applications. The GUI User Exits are provided as part of the GUI runtime installation, while the Block Mode User Exits are provided as part of the Implementation Toolset installation. The source code for all of the GUI User Exits can be found in **wrexitn.c**, while the Block Mode User Exits can be found in individual **tir*.c** source files. Batch files are provided to assist in building the User Exit DLL for each particular runtime environment which are discussed in the following sections.

User Exits Provided by COOL:Gen

COOL:Gen supplies the user exits in both source and object form. They are written in C.

The exits are located in the same directory as the COOL:Gen software.

You can use the exits as-is for default processing, or modify them to meet site-specific requirements.

Read the comment sections in any exit before modifying the exit. The comment sections provide specific information about each exit.

When the modifications are complete, compile the modified exit and install the new version. The process removes the existing version before replacing it with the newly built version, so save a backup copy of the existing version if you need it.

User exits also exist on MVS as part of Host Construction and on COOL:Gen Target systems. User exits are installed when Host Construction, Workstation Construction, or the Implementation Toolset components are installed. Although the default user exits work in a similar manner on most platforms, user exit processing can be customized so that it is specific to the computer on which it runs. User exits therefore *belong* to a specific computer and are not uploaded or moved to a target system with the load modules that use them. Specific processing might be limited by the constraints of the target operating system, the configuration of the compiler on which it runs, or the procedures of the site.

User Exits (GUI)

The following table summarizes the functions available through the User Exits for generated GUI applications.

User Exit Description	Name
Default Retry Limit Exit	WRDRTL
Ultimate Retry Limit Exit	WRURTL
Globalization Exit	WRGLB
System ID Exit	WRSYSID
User ID Exit	WRUSRID
Terminal ID Exit	WRTERMID

User Exit Description	Name
Uppercase Translation Exit	WRUPPR
Client Security Token User Exit	WRSECTOKEN
Client/Server Encryption Exit	WRSECENCRYPT
Client/Server Decryption Exit	WRSECDECRYPT
Client/Server Flow Failure Exit	WRSRVERROR
Preconnection User Exit	WRPRECONNECT
Postconnection User Exit	WRPOSTCONNECT
Predisconnection User Exit	WRPREDISC
Postdisconnection User Exit	WRPOSTDISC
Century Default Exit	WRDEFAULTYEAR
Client/Server Asynchronous Flow Server Failure Exit	WRASYNCSRVRERROR

GUI User Exits are rebuilt into the DLL WRE*vvv*N.DLL using the command procedure MKEXITSN.BAT. *vvv* refers to the version number. For example, 600 for release 6.00.

Preconnection User Exit (WRPRECONNECT)

Called prior to any connection to a database, typically to retrieve connection info from a custom interface, or to perform its own connection. This function can return values to suppress the default dialog that prompts for connection data and/or the COOL:Gen-provided connection to a database.

Postconnection User Exit (WRPOSTCONNECT)

Called after the connection to a database. Returns no values. Use this exit to invoke any external processes you want to perform at this time.

Predisconnection User Exit (WRPREDISC)

Called prior to any disconnect from a database, typically to perform its own disconnect. This function can return a value to suppress the COOL:Gen provided disconnect to a database.

Century Default Exit (WRDEFAULTYEAR)

WRDEFAULTYEAR() is invoked when a date or timestamp field receives input and the field's edit pattern specifies a 2-character year value. The current year and the input year are passed to WRDEFAULTYEAR(). By default, the current hundred year value is added to the 2-character year value and returned.

Postdisconnection User Exit (WRPOSTDISC)

Called after the disconnect from a database. Returns no values. Use this exit to invoke any external processes you want to perform at this time.

Default Retry Limit Exit (WRDRTL)

WRDRTL is the Default Retry Limit Exit. It allows the user to override the COOL:Gen-defined default value for the TRANSACTION RETRY LIMIT system attribute. TRANSACTION RETRY LIMIT is initialized to this value at the beginning of each new transaction. This value might subsequently be modified by a SET TRANSACTION RETRY LIMIT statement in an action diagram.

TRANSACTION RETRY LIMIT is used to specify the maximum number of times a transaction is to be retried when one of the following events occurs:

- A RETRY TRANSACTION action diagram statement executes.
- A deadlock or timeout occurs trying to access a database, and no WHEN DATABASE DEADLOCK OR TIMEOUT statement was provided for that entity action statement.

In these cases, any uncommitted database updates are rolled back, and an attempt will then be made to execute the application again. Once the number of retries, as indicated by the TRANSACTION RETRY COUNT system attribute, reaches TRANSACTION RETRY LIMIT or the value specified by the Ultimate Retry Limit Exit (see next section), no more retries can occur, and the application fails with a runtime error if the last retry attempt was unsuccessful.

If the Default Retry Limit Exit is *not* used, TRANSACTION RETRY LIMIT will be initialized to 10. If the Default Retry Limit Exit is used, it must not return a value greater than that specified in the Ultimate Retry Limit Exit. See the following section, Ultimate Retry Limit Exit (WRURTL) for more information.

Ultimate Retry Limit Exit (WRURTL)

WRURTL is the Ultimate Retry Limit Exit. It allows the user to specify a maximum value for the TRANSACTION RETRY LIMIT system attribute. This value can never be exceeded, either by a SET TRANSACTION RETRY LIMIT statement in an action diagram or by the Default Retry Limit Exit.

See the preceding section, Default Retry Limit Exit (WRDRTL), for an explanation of when and how the TRANSACTION RETRY LIMIT system attribute is used.

Once the number of retries, as indicated by the TRANSACTION RETRY COUNT system attribute, reaches TRANSACTION RETRY LIMIT or the value specified by the Ultimate Retry Limit Exit, no more retries can occur, and the application will fail with a runtime error if the last retry attempt was unsuccessful.

If the Ultimate Retry Limit Exit is not used, the maximum value of TRANSACTION RETRY LIMIT will be 99. The Ultimate Retry Limit Exit can be modified to return a value of zero to suppress all retry attempts for that environment.

Globalization Exit (WRGLB)

WRGLB is the globalization exit. It supplies to the GUI runtime information such as the numeric thousands separator character, the numeric decimal point character, the currency symbol, the date separator character, and the date order.

The numeric characters are used when editing numeric fields for output. The characters should correspond to the edit pattern in the model. For example, if the edit pattern in "@ZZZ.ZZZ,99", the WRGLB exit should specify "@" as the currency symbol, "." as the thousands separator, and "," as the decimal point.

Date and time information is used only for date and time fields where the model does not specify the edit pattern. In these cases, the GUI runtime uses this information to build a default edit pattern using the information provided by the WRGLB exit. For example, if the WRGLB exit specifies the date separator as "-" (a dash) and the date order is *yymmdd*, then the default date edit pattern is *yy-mm-dd*.

System ID Exit (WRSYSID)

WRSYSID is the COOL:Gen System ID Exit. It supplies the value for the LOCAL_SYSTEM_ID attribute. LOCAL_SYSTEM_ID can be placed on a window during window design. The value can be up to 8 characters in length. The default value supplied by the default version of WRSYSID is "WIN32."

User ID Exit (WRUSRID)

WRUSRID is the COOL:Gen User ID Exit. It supplies the value for the USER_ID attribute. USER_ID can be placed on a window during window design, and referenced by statements in a PrAD. The value can be up to 8 characters in length. The default value supplied by the default version of WRUSRID is "<NONE>."

Terminal ID Exit (WRTERMID)

WRTERMID is the COOL:Gen Terminal ID exit. It supplies the value for the TERMINAL_ID attribute. TERMINAL_ID can be placed on a window during window design, and referenced by statements in a PrAD. The value can be up to 8 characters in length. The default value supplied by the default version of WRTERMID is "<NONE>."

Uppercase Translation Exit (WRUPPR)

WRUPPR is the COOL:Gen Uppercase Translation Exit. This exit is called any time the runtime needs to use uppercase in a string. The default for Windows is to call the Windows function "AnsiUpper," which handles uppercase characters.

Client Security Token User Exit (WRSECTOKEN)

WRSECTOKEN is provided for COOL:Gen-generated client (DPC) applications.

WRSECTOKEN is called by the GUI Runtime component after a CCB is constructed to allow the DPC to select its security method. The Security Token returns to the DPC **one** of the following codes:

- SecurityUsed
- SecurityNotUsed
- SecurityError

SecurityUsed

If the Security Token return code is SecurityUsed, the GUI Runtime formats a security offset section made up of the Client User ID and Password system attributes, and the Security Token returned from the user exit.

The SecurityUsed flag (when parsed from the header) tells the Client Manager and Communications Bridge applications whether to retrieve the security data from the CCB security offset section.

SecurityNotUsed

If the Security Token return code is SecurityNotUsed, the security offset section is not added to the CCB and the security offset is set to zero. The UseSecurity field of the CCB is set to indicate the Client Manager or Communications Bridge should not retrieve security data from the CCB.

SecurityError

If the Security Token return code is SecurityError, it is returned to the Client and all cooperative processing is stopped.

Client/Server Encryption Exit (WRSECENCRYPT)

WRSECENCRYPT is the encryption user exit. It is provided for use with COOL:Gen-generated client (DPC) and COOL:Gen-generated server (DPS) applications.

The encryption user exit is called by the GUI Runtime when implemented in a DPC application, and by the Server Manager when implemented in a DPS application. The encryption user exit returns **one** of the following return codes:

- EncryptionUsed
- EncryptionNotUsed
- EncryptionFailure

Encryption User Exit Client Processing

If the Security Token user exit does not return an Error, the GUI Runtime calls the Encryption user exit. The Encryption user exit receives a pointer to the portion of the CCB containing the view data section and the security offset section. The Encryption user exit takes action based on the return code from the Security Token user exit.

Note: The user must supply compatible encryption and decryption routines for use with COOL:Gen Encryption and Decryption user exits.

SecurityUsed Return Code

If the Security Token user exit returned a SecurityUsed return code, and the user has implemented an encryption routine, the Encryption user exit encrypts the sensitive data in place, replacing the plain text data in the CCB with encrypted data. The Encryption user exit returns an EncryptionUsed return code and the GUI Runtime sets the EncryptedFlag in the CCB header to indicate the data is encrypted.

Encryption User Exit DPS Processing

The Server Manager calls the Encryption user exit before sending a fully formatted CCB to a DPC or to another DPS. The Encryption user exit takes action based on the return code from the Security Validation user exit.

SecurityNotUsed Return Code

If the Security Validation user exit returned a SecurityNotUsed return code, no encryption takes place and the Encryption user exit returns an EncryptionNotUsed return code. The CCB is not modified.

Security_Application_Error

If the Encryption user exit returns an error, it returns an Error Message and default return code, EncryptionNotUsed.

Client/Server Decryption Exit (WRSECDECRYPT)

WRSECDECRYPT is the decryption user exit. It is provided for use with COOL:Gen-generated client (DPC) and COOL:Gen-generated server (DPS) applications. It is also supplied with default implementation in the Client Manager and Communications Bridge applications.

The decryption user exit is called by the GUI Runtime when implemented in a DPC application and by the Server Manager when implemented in a DPS application. The DPC sends a fully formatted CCB to the Client Manager (CM) and returns one of the following return codes:

- DecryptionUsed
- DecryptionNotUsed
- DecryptionFailure

Decryption User Exit DPC Processing

The CM receives its input from a DPC through the CCB header and security offset section. The CCB header contains a UseSecurity Flag and an Encryption Flag. The security offset section contains the User ID and Password if provided by the DPC. The CM determines the need for security login data by parsing out the UseSecurity and EncryptionUsed flags from the CCB header.

UseSecurity Flag

The DPC has the option of telling the CM to use the User ID and Password stored in the security offset section of the CCB instead of using the configured or runtime server security parameters through the UseSecurity flag. The UseSecurity flag is returned from the Security Token user exit in the DPC and stored in the CCB header indicating CCB security should be used.

Encryption Flag

The DPC tells the CM that sensitive portions of the CCB header are encrypted using the encryption flag. This flag is returned from the Encryption user exit in the DPC and stored in the CCB header.

CM Processing

When the CM determines a user ID and password are required to service a cooperative request, it retrieves the UseSecurity and Encryption flags from the CCB header. If the DPC has set the UseSecurity flag to TRUE, the CM parses the user ID and password from the CCB security offset section.

If the Encryption flag is set to TRUE, the CM calls the Decryption Exit to decrypt the encrypted portions of the CCB. The CM then uses the user ID and password from the decrypted portions of the CCB to perform target-server login.

UseSecurity	Decryption	Then
True	True	Call Decryption User Exit before parsing security offset section for User ID and Password.
True	False	Parse security offset section for User ID and Password and log onto server.
False	Do Not Care	Get User ID and Password from Dialog

Decryption User Exit Errors

Any errors returned from the Decryption user exit or encountered while processing the user exit are returned to the DPC and the cooperative flow is aborted.

Note: The user must supply compatible encryption and decryption routines for use with COOL:Gen Encryption and Decryption user exits.

Decryption User Exit DPS Processing

When the DPS receives a CCB from a DPC, the DPS runtime parses out the CCB header EncryptedFlag which is saved in the UA area as the encrypted flag. This flag tells the DPS server manager that the sensitive data is encrypted.

The Server Manager calls the decryption user exit to decrypt the views of the data and the security information in the CCB. The decryption user exit returns a pointer to the decrypted data area, a long integer indicating the size of the decrypted area in bytes, a failure message, and **one** of the following return codes:

- DecryptionUsed
- DecryptionSizeExceeded
(error code)
- DecryptionNotUsed
(error code)

A server runtime error is sent to the DPC if the decryption user exit return code is not DecryptionUsed. DecryptionSizeExceeded and DecryptionNotUsed are error codes.

Default Decryption User Exit Stub

If no decryption routine is implemented by the user, the default user exit stub sets the return code to DecryptionNotUsed.

Client/Server Flow Failure Exit (WRSRVRERROR)

WRSRVRERROR is invoked when an error is detected during the processing of a synchronous client/server dialog flow initiated through the USE statement. This user exit allows the client to detect the error, analyze the problem and recover. The exit can override the standard error dialog, suppress it completely, and restart the client flow procedure. It is possible to customize the user exit to contain specific processing rules and conditions.

Client/Server Asynchronous Flow Server Failure Exit (WRASYNCSRVERROR)

WRASYNCSRVERROR is invoked when an error is detected during the processing of an asynchronous client/server flow. Behavior is similar to that of WRSRVRERROR

User Exits (Block Mode)

The following table summarizes the functions available through the User Exits for generated Block Mode applications.

Description of User Exit	**Name**
Decrypt Exit	TIRDCRYP
User Dialect Exit	TIRDLCT
Default Retry Limit Exit	TIRDRTL
Server Error Logging Exit	TIRELOG
Help Interface Exit	TIRHELP
MBCS Uppercase Translation Exit	TIRUPDB
Server to Server Error Exit	TIRSERRX
Message Table Exit	TIRMTQB
Encrypt Exit	TIRNCRYP
Customize DBMS Connection	TIROCONN.PC (Oracle) TIRCS95.SQC (MS SQL) TIRCODBC.C (ODBC) TIRDCONN.SQC (DB2) TIRSCONN.Q (Sybase) TIRXCONN.EC (Informix)
Security Interface Exit	TIRSECR
Security Validation Exit	TIRSECV
System ID Exit	TIRSYSID
User Termination Exit	TIRTERMA
Uppercase Translation Exit	TIRUPPR
Ultimate Retry Limit	TIRURTL
User ID Exit	TIRUSRID

Description of User Exit	Name
National Language Translation Exit	TIRXLAT
Process 2-digit date and set century	TIRYYX

Block Mode User Exits are rebuilt into the DLL's AECN.DLL (command line), AECTRNN.DLL (transaction based), AEETRNN.DLL (Encina) , AEMTRNN.DLL (MQ Series), and AEDTRNN.DLL (DCE) using the command procedures MKEXITS.BAT, MKXITENC.BAT, MKXITMQS.BAT, and MKXITDCE.BAT respectively.

Decrypt Exit (TIRDCRYP)

TIRDCRYP is the Decryption Exit used with COOL:Gen generated server applications. It is called to decrypt sensitive portions of a message as it enters the server application. Modification of TIRDCRYP allows the user to include their decryption algorithm. If the decryption is unsuccessful, then the failure message can be populated with an application message of choice to present to the client. The standard return code is DECRYPTION_NOT_USED, indicating that decryption was not performed.

User Dialect Exit (TIRDLCT)

TIRDLCT is the User Dialect exit. It supplies the current user’s dialect to the application. It is useful only for multilingual applications. TIRDLCT returns a dialect value of DEFAULT. For multilingual support, the user is responsible for modifying this module to return the appropriate dialect for a user. The dialect returned should be defined using the Design selection on the COOL:Gen action bar. If it is not, the default dialect for the application is used.

Default Retry Limit Exit (TIRDRTL)

The Default Retry Limit Exit lets you override the COOL:Gen-defined default value for the TRANSACTION RETRY LIMIT system attribute. TRANSACTION RETRY LIMIT will be initialized to this value at the beginning of each new transaction. This value can subsequently be modified by a SET TRANSACTION RETRY LIMIT statement in an action diagram.

TRANSACTION RETRY LIMIT is used to specify the maximum number of times a transaction is to be retried when **one** of the following events occurs:

- A RETRY TRANSACTION action diagram statement executes.
- A deadlock or timeout occurs trying to access a database, and no WHEN DATABASE DEADLOCK OR TIMEOUT statement was provided for that entity action statement.

In these cases, any uncommitted database updates will be rolled back, and an attempt will then be made to execute the application again. Once the number of retries, as indicated by the TRANSACTION RETRY COUNT system attribute, reaches TRANSACTION RETRY LIMIT or the value specified by the Ultimate Retry Limit Exit (see TIRURTL), no more retries can occur, and the application will fail with a runtime error if the last retry attempt was unsuccessful.

Default Retry Limit Exit Processing

If the Default Retry Limit Exit is not used, TRANSACTION RETRY LIMIT will be initialized to 10 for all target environments. If the Default Retry Limit Exit is used, it must not return a value greater than that specified in the Ultimate Retry Limit Exit (see TIRURTL).

Customizing the Default Retry Limit Exit

Copy TIRDRTL to one of your libraries or directories, and modify the copied exit as needed.

Server Error Logging Exit (TIRELOG)

TIRELOG is the COOL:Gen Server Error Logging Exit. It is called by the server error handling routine before the error response is returned to the client. It is called for COOL:Gen application-level errors (e.g., common format translation error). Modification of TIRELOG allows the user to customize server error logging and to create error tokens. The customized error tokens are passed to the client. The client error handling exit might be modified to customize how the error is handled. The default processing for TIRELOG is to return without logging or creating an error token.

Help Interface Exit (TIRHELP)

TIRHELP is the Help Interface Exit. It is called when a command of HELP or PROMPT is entered. From TIRHELP, a help system can be invoked to provide application help information.

The user exit handles these commands as shown in the following list.

- HELP command issued while the cursor is not on a field:

  ```
  No help available for this screen
  ```

- HELP command issued while the cursor is on a field:

  ```
  No help available for this field
  ```

- PROMPT command issued while the cursor is not on an enterable field:

  ```
  Prompt is valid only for enterable fields
  ```

- PROMPT command issued while the cursor is on an enterable field:

  ```
  No Prompt data available for this field
  ```

Code to invoke the application help system provided with COOL:Gen is included in this module as comments. The comments can be uncommented and used as an example of how to invoke a user-supplied help system.

(TIRUPDB)

TIRUPDB is the multibyte uppercase Translation Exit. The user can modify the mechanism used to uppercase multibyte text.

(TIRSERRX)

TIRSERRX is invoked when an error condition happens during server to server flows. The exit can influence the default runtime error behavior. When the NOTPROPAGATE_ERR is returned, the calling pstep continues the execution ignoring the fact that an error has occurred in the called pstep.

Message Table Exit (TIRMTQB)

TIRMTQB is the Runtime Error Message Table. It is used at application runtime when a failure is encountered and an error message is to be displayed.

TIRMTQB contains a table of all runtime error messages used by the runtime routines. The messages in this table are described in the Runtime Error Table. The user can change the wording of these messages.

Additional tables can also be defined for messages in other dialects to support multilingual applications. The table to be used for a given dialect is specified using the Design selection during model development.

Runtime Error Handling

Runtime errors are handled by the Dialog Manager. Runtime errors are non-fatal (screen edit) or fatal errors.

If a non-fatal error such as invalid user input occurs, the Dialog Manager displays an error message on the transaction screen. You can correct the error and continue processing the transaction.

If an application fails because of a fatal error, transaction processing is terminated, and the error processing is as follows:

- The Dialog Manager performs all necessary rollbacks of the database(s).
- COOL:Gen displays an error message screen that lists the appropriate runtime error message(s).
- Press Enter from the error message screen. COOL:Gen AE application terminates execution. ACMS applications will return to the screen that was displayed just before the error occurred.

Runtime Error Table

The Runtime Error Message Table includes an entry for each runtime error message. Each table entry includes the following information:

Message Type—Appears when a message number is not found in the table, or when you return to a transaction screen after a fatal error or a Dialog Manager error occurs. Valid message types are shown in the following list:

Default message—Appears when a message number is not found in the table, or when you return to a transaction screen after a fatal error or a Dialog Manager error occurs.

Dialog Manager error—Occurs when the Dialog Manager is unable to communicate with the system. This type of error is beyond the control of COOL:Gen and is fatal. An error in the load module packaging or in the configuration specifications causes a Dialog Manager error. Error handling is the same as for a fatal error.

Fatal error—A fatal error is a COOL:Gen application abnormal program ending. If a condition occurs at runtime that the generated code cannot handle, the system issues a fatal error. An error message screen displays the appropriate error message(s).

Function error—A function error occurs if a COOL:Gen-supported function receives invalid input or produces invalid output. (COOL:Gen-supplied functions manipulate characters, numbers, dates, and times.)

Screen edit error—A screen edit error is a non-fatal error that occurs when an input or output value for a field does not match the expected value (range, type, or format) defined for the field during model development. This type of message is displayed on your transaction screen. You can correct the error and continue with the transaction.

Unformatted input error—An unformatted input error occurs if the unformatted input contains invalid parameters, delimiters, or both. Unformatted input is a list of parameters associated with a clear screen transaction code.

Message Number—Each message has a unique number that is permanently assigned by COOL:Gen.

Message Text—The message text consists of the actual words that appear on the application screen when an error occurs. Because of the length of the message identifier, the message text is limited to 68 characters for an 80-character screen. The message text and any variables that can be appended are truncated if they exceed the length of the error message line defined for the application screen.

Suffix—(If applicable). The suffix contains variable values, such as return codes, permitted values, or the values in error.

Encrypt Exit (TIRNCRYP)

TIRNCRYP is the Encryption Exit used with COOL:Gen generated server applications. It is called to encrypt sensitive portions of a message before it leaves the server application. The portion of the message encrypted would be limited to the security information section and any sensitive data, for example, view data. Modification of TIRNCRYP allows the user to include their encryption algorithm. If the encryption is unsuccessful, then the failure message can be populated with an application message of choice to present to the client. The standard return code is ENCRYPTION_NOT_USED, indicating that encryption was not performed.

Customize DBMS Connection (TIROCONN.PC, TIRCS95.SQC, TIRCODBC.C, TIRDCONN.SQC, TIRSCONN.CP, TIRXCONN.EC)

Each module is the user exit that lets you customize the connection to the particular database.

Security Interface Exit (TIRSECR)

TIRSECR is the Security Interface Exit. It is used to perform transaction-level security. The default code for TIRSECR sets the return code to spaces, indicating no security violations.

The following data is provided by the Dialog Manager for use in checking security authorization:

- System ID
  (as provided by the System ID Exit)
- User ID
  (as provided by the User ID Exit)
- Trancode
- Terminal ID
- Load module name
- Procedure step name

If the security check passes, TIRSECR moves a value of spaces to the return code. If the security check fails, a non-blank value is moved to the return code with a message describing the violation.

When the Dialog Manager receives control, it proceeds with the transaction if the return code is spaces, or issues an error if it is not.

Security Validation Exit (TIRSECV)

TIRSECV is the security validation user exit. This user exit lets you implement external security packages in the server environment to verify the Client User ID and Client Password parsed from the security section of the CCB.

TIRSECV is called by the server manager if the security offset section exists in the CCB received from the DPC. TIRSECV parses the security section to get the Client User ID, Client Password, and Security Token sent by the DPC.

TIRSECV receives as input the following parameters:

ClientUserid—Pointer to Client User ID

ClientPassword—Pointer to Client Password

PSecurityToken—Pointer to Security Token

ISecurityTokenLen—Long integer indicating the length of the Security Token

Trancode—Pointer to the Trancode

TIRSECV returns:

- Failure Message
- Return Code indicating
  - SecurityUsed
  - SecurityNotUsed
  - Security_Application_Error

If the ReturnCode is not SecurityUsed, then a server runtime execution type error buffer is sent to the client. If TIRSECV is unable to translate the security section within the CCB, a server runtime Dialog type error buffer is sent to the DPC. TIRSECW copies the CCB Security Token to the GURB-REST temporary buffer and sets the pointer to this address.

Note: The user must provide a security validation routine for use with the Security Validation Exit.

Default TIRSECV User Exit Stub

If no security verification routine is implemented by the user, the default user exit stub sets the return code to SECURITY_NOT_USED.

System ID Exit (TIRSYSID)

TIRSYSID is the System ID Exit. It supplies the system ID to the application. TIRSYSID calls runtime routine DEFSYSID. This routine returns a default system ID, the value of which depends on the platform on which the system is executing. The purpose of TIRSYSID is to implement COOL:Gen logic which lets you implement one model on multiple platforms, and perform processing appropriate for the platform.

User Termination Exit (TIRTERMA)

TIRTERMA is the User Termination exit. It is called when an application fails. Modification of TIRTERMA allows the user to customize the handling of runtime errors. The default processing for TIRTERMA returns a status code of spaces, indicating that standard error handling should be used.

Uppercase Translation Exit (TIRUPPR)

TIRUPPR is the Uppercase Translation Exit. It contains tables used to translate input from lower to upper case. TIRUPPR contains one table named DEFAULT for the English character set. For other languages, this table can be expanded or new tables can be added. The table name to be used for a given dialect is specified using the Design selection on the COOL:Gen action bar.

Ultimate Retry Limit (TIRURTL)

The Ultimate Retry Limit Exit allows your to specify a maximum value for the TRANSACTION RETRY LIMIT system attribute. This value can never be exceeded, by a SET TRANSACTION RETRY LIMIT statement in an action diagram, or by the Default Retry Limit Exit.

See the Default Retry Limit Exit (WRDRTL) section in this guide for an explanation of when and how the TRANSACTION RETRY LIMIT system attribute is used.

Once the number of retries, as indicated by the TRANSACTION RETRY COUNT system attribute, reaches either TRANSACTION RETRY LIMIT or the value specified by the Ultimate Retry Limit Exit, no more retries can occur, and the application will fail with a runtime error if the last retry attempt was unsuccessful.

Ultimate Retry Limit Exit Processing

If the Ultimate Retry Limit Exit is not used, the maximum value of TRANSACTION RETRY LIMIT will be 99 for all target environments. The Ultimate Retry Limit Exit can be modified to return a value of zero to suppress all retry attempts for that environment.

Customizing the Ultimate Retry Limit Exit

Copy TIRURTL to one of your libraries or directories and modify as necessary.

User ID Exit (TIRUSRID)

TIRUSRID is the User ID exit. It is used to supply the user's ID to the application. TIRUSRID calls runtime routine DEFUSRID. This routine returns a default user ID, the value of which depends on the platform on which the system is executing.

National Language Translation Exit (TIRXLAT)

TIRXLAT allows the conversion of textual data based on from/to codepage and operating system information. If a suitable translation table is not found, the data will be passed back without translation. The user can replace a translation table to customize their environment.

Process 2-Digit Date and Set Century (TIRYYX)

TIRYYX is used to process two-digit or *yy*-style date input and to set the century part using any fixed-window, sliding-window or other algorithm of choice, when using COOL:Gen in the standard map generation mode.

Internally, COOL:Gen handles 4-digit year dates correctly assuming the user application uses a *yyyy* edit pattern throughout. If the user interface is designed to accept a two-digit date entry, and defaulting to the current century is not acceptable, use this exit to implement logic to get the desired behavior for defaulting the century part of the date.

Default Exit Processing

The default exit does not perform any processing. COOL:Gen defaults are returned

Customizing the Exit

Copy the default exit from the COOL:Gen sample library to a separate library. The member name is TIRYYX.

Modify the exit to perform the specific desired functions using the instructions in the exit source code file.

When you have finished your modifications, install and test the exit.

Appendix C

Troubleshooting

Since troubleshooting information can vary based on your environment, this appendix contains troubleshooting information for each Build Tool environment.

Problems and Resolutions

The following lists various problems that might occur when using the Build Tool with Windows NT. Follow the recommendations provided in the resolutions section of the table to recover normal operation.

Stream of Warnings

Problem	A stream of warnings appears whenever a model is built.
Resolution	Typically, only errors or lines preceded by multiple asterisks (*** ...) indicate a valid problem when the results of a a build are reviewed. Watching the session during a build might provide indications of any errors that are encountered.

Warning levels in the Review output file are enabled so that you can return to the list in case any problems are seen when running the COOL:Gen application.

The following list contains typical warnings which can be expected at the onset of Review. These warnings are **not** significant.

- `...warning C4018:‘initializing’: signed/un-signed mismatch`
- `...warning C4035:‘x’:no return value`
- `...warning C4101:‘x’:unreferenced local variable`
- `...warning C4761: integral size mismatch in argument; conversion supplied`
- `...warning C4013:‘x’:undefined;assuming extern returning int`
- `...warning C4051: type conversion; possible loss of data`

If you prefer that the compiler not display warning messages, then make the following modifications to the INTDBCRI.SCR and INTDBCLM.SCR scripts:

1. Make a backup copy of both scripts.
2. Edit both scripts to change "–W3" to "–W0".

Error - L1093: ... object not found

Problem

Reviewing an RI or LM module build reveals the following error:

```
L1093: .. object not found.
```

Resolution

Perform the following steps:

1. Verify that the SQL precompiler has enough memory and that the referenced object file exists.
2. Rebuild the module.

Failed Build that Seemed Successful

Problem

You receive indications of a successful build when the build has obviously failed.

Resolution

The following are reasons why this might have occurred:

1. The batch file that performs the module build has no means by which it can feed status information back to the Build Tool window. The generic message:

   ```
   Review results when Build is done.
   ```

   displays instead, regardless of whether the build was successful.
2. Batch files detect errors from ERRORLEVEL settings that a program may or may not set. NT itself, when displaying an error message like:

   ```
   Bad command or filename.
   ```

   does not set the ERRORLEVEL to a non-zero value, so the batch file is unable to detect system problems. Oracle utilities, compiler and linker utilities might not set the ERRORLEVEL either, so the batch file does not know that errors were encountered. When a module build fails, watch the session for any telltale standard errors to more accurately determine the level of success of the build.
3. Related to the previous explanation, NT does not provide the command line with the ability for batch files to capture messages to STDERR; only to STDOUT. In other words, the batch file can trap information or warning messages and funnel them to the output review file, but error messages sent to the STDOUT Input/Output stream are not captured. This is why rebuilding a module and viewing the build session will normally provide accurate clues to any errors that might be encountered.

Compile Step Failed #1

Problem

The compile step fails and yields an error such as “Bad command or filename.”

Resolution

Use an NT Command Prompt and perform the following checks:

1. Enter **CL** at the prompt to check compiler availability.
2. Enter **LINK** at the prompt to check Linker availability.
3. Enter **RC** at the prompt to check GUI Resource Compiler availability.

Models Will Not Build

Problem

Models will not build.

Resolution

Perform the following steps:

1. Verify your system configuration and ensure that Microsoft Visual C++ is the Windows NT compiler.
2. Close all other Windows applications or restart Windows to optimize available resources for the current environment.

Note: A network connection is not required during LM and RI module builds.

Load Module Failed to Generate

Problem

Load module fails to generate, displays the following message:

```
Precompile on P6052435.SQC failed for [model_name].
```

Resolution

This is an indication that certain parameters for the current user might not have been properly set.

1. Using any standard text editor, open the INTDBCLM.SCR initialization file.
2. Change the IRECLEN= and ORECLEN= of the PROC command to be large enough for the longest input line. We recommend a value of 260.
3. Reinitialize your system for the changes to take effect.

Compile Step Failed #2

Problem	The compile step fails during the building of a module.
Resolution	This is normally an indication that your workstation is not properly configured. See the software requirements in the *Technical Requirements* sheet and verify that all the necessary software components have been properly built. If it becomes necessary to rebuild one or more of the items on the list, see the *Installing COOL:Gen* guide. Close all other Windows NT applications or restart Windows NT, then rebuild the module. It is likely that a support application is unavailable or not properly configured in the PATH.

Consistency Check Failed

Problem	Consistency check fails saying that functions are available only under NT, but NT is the actual platform.
Resolution	Text manipulation file functions such as CREATETEXTFILE are available only to GUI windowed procedures. Make sure your action blocks have windows for these functions.

Error - Can't Find MSVCRT40.DLL

Problem

You receive the following error when running a COOL:Gen application on a production workstation:

```
Can't Find MSVCRT40.DLL
```

Resolution

When an application is constructed with any version of MSVC, the appropriate MSVCRT*xx*.DLL must also be distributed as the application is distributed to other production workstations.

Index