Pro*C/C++

Getting Started

10*g* Release 2 (10.2) for Microsoft Windows (32-Bit)

B14321-01

June 2005

ORACLE®

Pro*C/C++ Getting Started, 10*g* Release 2 (10.2) for Microsoft Windows (32-Bit)

B14321-01

Contents

List of Tables

Preface

This manual provides introductory information for the Pro*C/C++ precompiler running on Windows operating systems.

This Preface contains these topics:

Intended Audience

*Pro*C/C++ Getting Started* is intended for anyone who wants to use Pro*C/C++ to perform the following tasks:

- Embed SQL statements in a C or C++ program.
- Build Oracle database applications with Pro*C/C++.

To use this document, you need to know:

- Commands for deleting and copying files and the concepts of the search path, subdirectories, and path names.
- How to use the Windows operating system.
- Visual C++ version 5.0 or higher.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

For more information, see these Oracle resources:

- *Oracle Database Installation Guide for 32-Bit Windows*
- *Oracle Database Release Notes for Windows*
- *Pro*C/C++ Programmer's Guide*
- *Oracle Database Platform Guide for Windows*
- *Oracle Enterprise Manager Administrator's Guide*
- *Oracle Database Net Services Administrator's Guide*
- *Oracle Real Application Clusters Quick Start*
- *Oracle Database New Features*
- *Oracle Database Concepts*
- *Oracle Database Reference*
- *Oracle Database Error Messages*

Many of the books in the documentation library use the sample schemas of the seed database, which is installed by default when you install Oracle. Refer to *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in Pro*C/C++?

This section describes new features of Oracle Database 10*g* releases and provides pointers to additional information. New features information from previous releases is also retained to help those users migrating to the current release.

The following sections describe the new features in Oracle Pro*C/C++:

Oracle Database10*g* Release 2 (10.2) New Features in Pro*C/C++

There is no new Windows specific feature in Pro*C/C++ for this release.

Oracle9*i* Release 2 (9.2) New Features in Pro*C/C++

There is no new Windows specific feature in Pro*C/C++ for this release.

See Also: "What's New" preface of *Pro*C/C++ Programmer's Guide*

Oracle9*i* Release 1 (9.0.1) New Features in Pro*C/C++

The Oracle9*i* release 1 (9.0.1) feature described in this section highlights the support for Windows 2000.

Using Oracle9*i* on Windows 2000

Pro*C/C++ is now supported on Windows 2000. There are some differences between using Oracle9*i* on Windows 2000.

See Also: *Oracle Database Platform Guide for Windows*

Oracle8*i* Release 8.1.6 New Features in Pro*C/C++

The Oracle8*i* release 8.1.6 features and enhancements described in this section comprise the overall effort to make Pro*C/C++ application development simpler.

Fully Integrated Debugging Capabilities

Beginning with release 8.1.6, the behavior of the LINES={YES|NO} option has changed. Now, when LINES=YES is specified, a #line preprocessor directive is generated after every line of generated code in the output program. This enables developers using debuggers such as GDB or IDEs such as the Microsoft Visual Studio for C++ to debug their application programs by viewing the Pro*C/C++ source program instead of by stepping through the generated code.

> **See Also:** Integrating Pro*C/C++ within Microsoft Visual C++ Projects

1

Introducing Pro*C/C++

This chapter describes Pro*C/C++, the Oracle programmatic interface for the C and C++ languages running on Windows operating systems. Pro*C/C++ enables you to build Oracle database applications in a Win32 environment.

This chapter contains these topics:

> **See Also:** *Pro*C/C++ Programmer's Guide* for additional information

What Is Pro*C/C++?

The Pro*C/C++ precompiler enables you to create applications that access your Oracle database whenever rapid development and compatibility with other systems are your priorities.

The Pro*C/C++ programming tool enables you to embed Structured Query Language (SQL) statements in a C or C++ program. The Pro*C/C++ precompiler translates these statements into standard Oracle runtime library calls, then generates a modified source program that you can compile, link, and run in the usual way.

Features

Pro*C/C++ supports the following features:

- Remote access with Oracle Net Services or local access to Oracle databases
- Embedded PL/SQL blocks
- Bundled database calls, which can provide better performance in client/server environments
- Full ANSI compliance for embedded SQL programming
- PL/SQL version 9.0 and host language arrays in PL/SQL procedures
- Multi-threaded applications
- Full ANSI C compliance

- Can be deployed in Instant Client environments. For more information, refer to the OCI Instant Client documentation.
- Microsoft Visual C++ support, version 6.0 for 32-bit applications

Note: Borland C++ is no longer supported.

Restrictions

Pro*C/C++ does not support 16-bit code generation.

Directory Structure

Installing Oracle software creates a directory structure on your hard drive for the Oracle products. A main Oracle directory contains the Oracle subdirectories and files that are necessary to run Pro*C/C++.

When you install Pro*C/C++, Oracle Universal Installer creates a directory called `\precomp` in the *`ORACLE_BASE`*`\`*`ORACLE_HOME`* directory. This subdirectory contains the Pro*C/C++ executable files, library files, and sample programs listed in Table 1–1.

Table 1–1 precomp Directory Structure

Directory Name	Contents
`\admin`	Configuration files
`\demo\proc`	Sample programs for Pro*C/C++
`\demo\sql`	SQL scripts for sample programs
`\doc\proc`	Readme files for Pro*C/C++
`\help\proc`	Help files for Pro*C/C++
`\lib\msvc`	Library files for Pro*C/C++
`\mesg`	Message files
`\misc\proc`	Miscellaneous files for Pro*C/C++
`\public`	Header files

Note: The `\precomp` directory can contain files for other products, such as Pro*COBOL.

Known Problems, Restrictions, and Workarounds

Although all Windows operating systems allow spaces in file names and directory names, the Oracle Pro*C/C++ and Oracle Pro*COBOL precompilers will not precompile files that include spaces in the filename or directory name. For example, do not use the following formats:

- `proc iname=test one.pc`
- `proc iname=d:\dir1\second dir\sample1.pc`

2

Using Pro*C/C++

This chapter explains how to create and precompile a project. It also explains how to use Pro*C/C++ at the command prompt.

This chapter contains these topics:

- Header Files
- Library Files
- Multithreaded Applications
- Precompiler Options
- Using Pro*C/C++ with the Oracle XA Library

> **See Also:** *Pro*C/C++ Programmer's Guide* for additional information

Using Pro*C/C++ at the Command Prompt

To precompile a file at the command prompt, enter the following command:

```
C:\> proc iname=filename.pc
```

where *filename*.pc is the name of the file. If the file is not in your current working directory, include the file's full path after the INAME argument.

Pro*C/C++ generates *filename*.c, which can be compiled by your C compiler.

Header Files

The *ORACLE_BASE**ORACLE_HOME*\precomp\public directory contains the Pro*C/C++ header files. Table 2–1 lists and describes the header files.

> **See Also:** *Pro*C/C++ Programmer's Guide* for more information about oraca.h, sqlca.h, and sqlda.h.

Table 2–1 Header Files

Header Files	Description
`oraca.h`	Contains the Oracle Communications Area (ORACA), which helps you to diagnose runtime errors and to monitor your program's use of various Oracle Database 10*g* resources.
`sql2oci.h`	Contains SQLLIB functions that enable the Oracle Call Interface (OCI) environment handle and OCI service context to be obtained in a Pro*C/C++ application.
`sqlapr.h`	Contains ANSI prototypes for externalized functions that can be used in conjunction with OCI.
`sqlca.h`	Contains the SQL Communications Area (SQLCA), which helps you to diagnose runtime errors. The SQLCA is updated after every executable SQL statement.
`sqlcpr.h`	Contains platform-specific ANSI prototypes for SQLLIB functions that are generated by Pro*C/C++. By default, Pro*C/C++ does not support full-function prototyping of SQL programming calls. If you need this feature, include `sqlcpr.h` before any EXEC SQL statements in your application source file.
`oraca.h`	Contains the Oracle Communications Area (ORACA), which helps you to diagnose runtime errors and to monitor your program's use of various Oracle Database 10*g* resources.
`sql2oci.h`	Contains SQLLIB functions that enable the Oracle Call Interface (OCI) environment handle and OCI service context to be obtained in a Pro*C/C++ application.
`sqlapr.h`	Contains ANSI prototypes for externalized functions that can be used in conjunction with OCI.

Library Files

The *ORACLE_BASE*`\`*ORACLE_HOME*`\precomp\lib\msvc` directory contains the library file that you use when linking Pro*C/C++ applications. The library file is called `orasql110.lib`.

Pro*C/C++ application program interface (API) calls are implemented in DLL files provided with your Pro*C/C++ software. To use the DLLs, you must link your application with the import libraries (.lib files) that correspond to the Pro*C/C++ DLLs. Also, you must ensure that the DLL files are installed on the computer that is running your Pro*C/C++ application.

Microsoft provides you with three libraries: `libc.lib`, `libcmt.lib`, and `msvcrt.lib`. The Oracle DLLs use the `msvcrt.lib` runtime library. You must link with `msvcrt.lib` rather than the other two Microsoft libraries.

Multithreaded Applications

Build multithreaded applications if you are planning to perform concurrent database operations.

Windows 2000, and Windows 98 schedule and allocate threads belonging to processes. A thread is a path of a program's execution. It consists of a kernel stack, the state of the CPU registers, a thread environment block, and a users stack. Each thread shares the resources of a process. Multithreaded applications use the resources of a process to coordinate the activities of individual threads.

When building a multithreaded application, make sure that your C/C++ code is reentrant. This means that access to static or global data must be restricted to one

thread at a time. If you mix multithreaded and non-reentrant functions, one thread can modify information that is required by another thread.

The Pro*C/C++ precompiler automatically creates variables on the local stack of the thread. This ensures that each thread using the Pro*C/C++ function has access to a unique set of variables and is reentrant.

> **See Also:** *Pro*C/C++ Programmer's Guide*for additional information on how to write multithreaded applications with Pro*C/C++

Precompiler Options

This section highlights issues related to Pro*C/C++ for Windows platforms.

> **See Also:** "Precompiler Options" of *Pro*C/C++ Programmer's Guide*

Configuration File

A configuration file is a text file that contains precompiler options.

For this release, the system configuration file is called `pcscfg.cfg`. This file is located in the *`ORACLE_BASE`*`\`*`ORACLE_HOME`*`\precomp\admin` directory.

CODE

The `CODE` option has a default setting of `ANSI_C`. Pro*C/C++ for other operating systems may have a default setting of `KR_C`.

DBMS

`DBMS=V6_CHAR` is not supported when using `CHAR_MAP=VARCHAR2`. Instead, use `DBMS=V7`.

INCLUDE

For sample programs that precompile with `PARSE=PARTIAL` or `PARSE=FULL`, an include path of `c:\program files\devstudio\vc\include` has been added. If Microsoft Visual C++ has been installed in a different location, modify the Include Directories field accordingly for the sample programs to precompile correctly.

PARSE

The `PARSE` option has a default setting of `NONE`. Pro*C/C++ for other operating systems may have a default setting of `FULL`.

Using Pro*C/C++ with the Oracle XA Library

The XA Application Program Interface (API) is typically used to enable an Oracle database to interact with a Transaction Processing (TP) monitor, such as:

- BEA Tuxedo
- IBM Transarc Encina
- IBM CICS

You can also use TP monitor statements in your client programs. The use of the XA API is also supported from both Pro*C/C++ and OCI.

The Oracle XA Library is automatically installed as part of Oracle Database 10*g* Enterprise Edition. The following components are created in your Oracle home directory:

Table 2–2 Oracle XA Library Components and Locations

Component	Location
`oraxa9.lib`	*ORACLE_BASE*`\`*ORACLE_HOME*`\rdbms\xa`
`xa.h`	*ORACLE_BASE*`\`*ORACLE_HOME*`\rdbms\demo`

Compiling and Linking a Pro*C/C++ Program with XA

To compile and link a Pro*C/C++ program with XA:

1. Precompile *`filename`*`.pc` using Pro*C/C++ to generate *`filename`*`.c`.
2. Compile *`filename`*`.c`, making sure to include *ORACLE_BASE*`\`*ORACLE_HOME*`\rdbms\xa` in your path.
3. Link *`filename`*`.obj` with the following libraries:

Library	Location
`oraxa10.lib`	*ORACLE_BASE*`\`*ORACLE_HOME*`\rdbms\xa`
`oci.lib`	*ORACLE_BASE*`\`*ORACLE_HOME*`\oci\lib\msvc`
`orasql10.lib`	*ORACLE_BASE*`\`*ORACLE_HOME*`\precomp\lib\msvc`

1. Run *`filename`*`.exe`.

XA Dynamic Registration

Oracle supports the use of XA dynamic registration. XA dynamic registration improves the performance of applications that interface with XA-compliant TP monitors.

For TP monitors to use XA dynamic registration with an Oracle database, you must add either an environmental variable or a registry variable to the Windows computer on which your TP monitor is running. See either of the following sections for instructions:

- Adding an Environmental Variable for the Current Session
- Adding a Registry Variable for All Sessions

Adding an Environmental Variable for the Current Session

Adding an environmental variable at the command prompt affects only the current session.

To add an environmental variable for the current session:

1. Go to the computer where your TP monitor is installed.
2. Enter the following at the command prompt:

```
C:\> set ORA_XA_REG_DLL = vendor.dll
```

where *vendor*.dll is the TP monitor DLL provided by your vendor.

Adding a Registry Variable for All Sessions

Adding a registry variable affects all sessions on your Windows computer. This is useful for computers where only one TP monitor is running.

To add a registry variable for all sessions:

1. Go to the computer where your TP monitor is installed.
2. Enter the following at the command prompt:

   ```
   C:\> regedt32
   ```

   The Registry Editor window appears.
3. Go to HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\HOME*ID*.
4. Select the Add Value option in the Edit menu.

   The Add Value dialog box appears.
5. Enter ORA_XA_REG_DLL in the Value Name field.
6. Select REG_EXPAND_SZ from the Data Type list .
7. Click **OK**.

   The String Editor dialog appears.
8. Enter *vendor*.dll in the String field, where *vendor*.dll is the TP monitor DLL provided by your vendor.
9. Click **OK**.

   The Registry Editor adds the parameter.
10. Select Exit from the Registry menu.

    The registry exits.

XA and TP Monitor Information

Refer to the following for more information about XA and TP monitors:

- *Distributed TP: The XA Specification* (C193) published by the Open Group. See the Web site at:

  ```
  http://www.opengroup.org/publications/catalog/tp.htm
  ```

- The Open Group., 1010 El Camino Real, Suite 380, Menlo Park, CA 94025, U.S.A.
- Your specific TP monitor documentation

> **See Also:** *Oracle Database Application Developer's Guide - Fundamentals*for more information about the Oracle XA Library and using XA dynamic registration

3

Sample Programs

This chapter describes how to build Oracle database applications with Pro*C/C++ using the sample programs that are included with this release.

This chapter contains these topics:

- Sample Program Descriptions
- Building the Demonstration Tables
- Building the Sample Programs

Sample Program Descriptions

When you install Pro*C/C++, Oracle Universal Installer copies a set of Pro*C/C++ sample programs to the *ORACLE_BASE**ORACLE_HOME*\precomp\demo\proc directory. These sample programs are listed in Table 3–1, " Sample Programs" and described in the subsequent section.

When built, the sample programs that Oracle provides produce `.exe` executables.

For some sample programs, as indicated in the Notes column of the table, you must run the SQL scripts in the sample directory before you precompile and run the sample program. The SQL scripts set up the correct tables and data so that the sample programs run correctly. These SQL scripts are located in the *ORACLE_BASE**ORACLE_HOME*\precomp\demo\sql directory.

Oracle Corporation recommends that you build and run these sample programs to verify that Pro*C/C++ has been installed successfully and operates correctly. You can delete the programs after you use them.

You can build the sample program using a batch file called `pcmake.bat` or using Visual C++ 6.0.

See Also: "Building the Sample Programs" on page 3-6

Table 3–1 Sample Programs

Sample Program	Source Files	Pro*C/C++ GUI Project File	MSVC Compiler Project File	Notes
ANSIDYN1	ansidyn1.pc	ansidyn1.pre	ansidyn1.dsp	-
ANSIDYN2	ansidyn2.pc	ansidyn2.pre	ansidyn2.dsp	-
COLDEMO1	coldemo1.h coldemo1.pc coldemo1.sql coldemo1.typ	coldemo1.pre	coldemo1.dsp	Run coldemo1.sql and the Object Type Translator before building coldemo1.
CPDEMO1	cpdemo1.pc	cpdemo1.pre	cpdemo1.dsp	-

Table 3–1 (Cont.) Sample Programs

Sample Program	Source Files	Pro*C/C++ GUI Project File	MSVC Compiler Project File	Notes
CPDEMO2	cpdemo2.pc	cpdemo2.pre	cpdemo2.dsp	-
CPPDEMO1	cppdemo1.pc	cppdemo1.pre	cppdemo1.dsp	-
CPPDEMO2	cppdemo2.pc empclass.pc cppdemo2.sql empclass.h	cppdemo2.pre	cppdemo2.dsp	Run cppdemo2.sql before building cppdemo2.
CPPDEMO3	cppdemo3.pc	cppdemo3.pre	cppdemo3.dsp	-
CVDEMO	cv_demo.pc cv_demo.sql	cv_demo.pre	cv_demo.dsp	Run cv_demo.sql before building cv_demo.
EMPCLASS	cppdemo2.pc empclass.pc cppdemo2.sql empclass.h	empclass.pre	empclass.dsp	Run cppdemo2.sql before building empclass.
LOBDEMO1	lobdemo1.h lobdemo1.pc lobdemo1.sql	lobdemo1.pre	lobdemo1.dsp	Run lobdemo1.sql before building lobdemo1.
MLTTHRD1	mltthrd1.pc mltthrd1.sql	mltthrd1.pre	mltthrd1.dsp	Run mltthrd1.sql before building mltthrd1.
NAVDEMO1	navdemo1.h navdemo1.pc navdemo1.sql navdemo1.typ	navdemo1.pre	navdemo1.dsp	Run navdemo1.sql and the Object Type Translator before building navdemo1.
OBJDEMO1	objdemo1.h objdemo1.pc objdemo1.sql objdemo1.typ	objdemo1.pre	objdemo1.dsp	Run objdemo1.sql and the Object Type Translator before building objdemo1.
ORACA	oraca.pc oracatst.sql	oraca.pre	oraca.dsp	Run oracatst.sql before building oraca.
PLSSAM	plssam.pc	plssam.pre	plssam.dsp	-
SAMPLE	sample.pc	sample.pre	sample.dsp	-
SAMPLE1	sample1.pc	sample1.pre	sample1.dsp	-
SAMPLE2	sample2.pc	sample2.pre	sample2.dsp	-
SAMPLE3	sample3.pc	sample3.pre	sample3.dsp	-
SAMPLE4	sample4.pc	sample4.pre	sample4.dsp	-
SAMPLE5	sample5.pc exampbld.sql examplod.sql	sample5.pre	sample5.dsp	Run exampbld.sql, then run examplod.sql, before building sample5.
SAMPLE6	sample6.pc	sample6.pre	sample6.dsp	-
SAMPLE7	sample7.pc	sample7.pre	sample7.dsp	-
SAMPLE8	sample8.pc	sample8.pre	sample8.dsp	-
SAMPLE9	sample9.pc calldemo.sql	sample9.pre	sample9.dsp	Run calldemo.sql before building sample9.
SAMPLE10	sample10.pc	sample10.pre	sample10.dsp	-
SAMPLE11	sample11.pc sample11.sql	sample11.pre	sample11.dsp	Run sample11.sql before building sample11.
SAMPLE12	sample12.pc	sample12.pre	sample12.dsp	-
SCDEMO1	scdemo1.pc	scdemo1.pre	scdemo1.dsp	-

Table 3–1 (Cont.) Sample Programs

Sample Program	Source Files	Pro*C/C++ GUI Project File	MSVC Compiler Project File	Notes
SCDEMO2	scdemo2.pc	scdemo2.pre	scdemo2.dsp	-
SQLVCP	sqlvcp.pc	sqlvcp.pre	sqlvcp.dsp	-
WINSAM	resource.h winsam.h winsam.ico winsam.pc winsam.rc	winsam.pre	winsam.dsp	-

The following subsections describe the functionality of the sample programs.

ANSIDYN1
Demonstrates using ANSI dynamic SQL to process SQL statements that are not known until runtime. This program is intended to demonstrate the simplest (though not the most efficient) approach to using ANSI dynamic SQL.

ANSIDYN2
Demonstrates using ANSI dynamic SQL to process SQL statements that are not known until runtime. This program uses the Oracle extensions for batch processing and reference semantics.

COLDEMO1
Fetches census information for California counties. This program demonstrates various ways to navigate through collection-typed database columns.

CPDEMO1
Demonstrates how the connection pool feature can be used. It also shows how different connection pool options can be used to optimize performance.

CPDEMO2
Demonstrates connection pool feature with relatively complex set of SQL statements and shows how performance gain depends on the kind of SQL statements used by the program.

CPPDEMO1
Prompts the user for an employee number, then queries the `emp` table for the employee's name, salary, and commission. This program uses indicator variables (in an indicator struct) to determine whether the commission is `NULL`.

CPPDEMO2
Retrieves the names of all employees in a given department from the `emp` table (dynamic SQL Method 3).

CPPDEMO3
Finds all salespeople and prints their names and total earnings (including commissions). This program is an example of C++ inheritance.

CVDEMO
Declares and opens a ref cursor.

EMPCLASS
The `EMPCLASS` and `CPPDEMO2` files were written to provide an example of how to write Pro*C/C++ programs within a C++ framework. `EMPCLASS` encapsulates a specific query on the `emp` table and is implemented using a cursor variable. `EMPCLASS` instantiates an instance of that query and provides cursor variable functionality (that

is: open, fetch, close) through C++ member functions that belong to the emp class. The empclass.pc file is *not* a standalone demo program. It was written to be used by the cppdemo2 demo program. To use the emp class, you have to write a driver (cppdemo2.pc) which declares an instance of the emp class and issues calls to the member functions of that class.

LOBDEMO1
Fetches and adds crime records to the database based on the person's Social Security Number. This program demonstrates the mechanisms for accessing and storing large objects (LOBs) to tables and manipulating LOBs through the stored procedures available through the DBMS_LOB package.

MLTTHRD1
Shows how to use threading in conjunction with precompilers. The program creates as many sessions as there are threads.

See Also: "Multithreaded Applications" on page 2-2

NAVDEMO1
Demonstrates navigational access to objects in the object cache.

OBJDEMO1
Demonstrates the use of objects. This program manipulates the object types *person* and *address*.

ORACA
Demonstrates how to use ORACA to determine various performance parameters at runtime.

PLSSAM
Demonstrates the use of embedded PL/SQL blocks. This program prompts you for an employee name that already resides in a database. It then executes a PL/SQL block, which returns the results of four SELECT statements.

SAMPLE
Adds new employee records to the personnel database and checks database integrity. The employee numbers in the database are automatically selected using the current maximum employee number +10.

SAMPLE1
Logs on to an Oracle database, prompts the user for an employee number, queries the database for the employee's name, salary, and commission, and displays the result. The program continues until the user enters 0 as the employee number.

SAMPLE2
Logs on to an Oracle database, declares and opens a cursor, fetches the names, salaries, and commissions of all salespeople, displays the results, and closes the cursor.

SAMPLE3
Logs on to an Oracle database, declares and opens a cursor, fetches in batches using arrays, and prints the results using the print_rows() function.

SAMPLE4
Demonstrates the use of type equivalencies using the LONG VARRAW external datatype.

SAMPLE5
Prompts the user for an account number and a debit amount. The program verifies that the account number is valid and that there are sufficient funds to cover the withdrawal before it debits the account. This program shows the use of embedded SQL.

SAMPLE6
Creates a table, inserts a row, commits insert, and drops the table (dynamic SQL Method 1).

SAMPLE7
Inserts two rows into the `emp` table and deletes them (dynamic SQL Method 2).

SAMPLE8
Retrieves the names of all employees in a given department from the `emp` table (dynamic SQL Method 3).

SAMPLE9
Connects to an Oracle database using the `scott/tiger` account. The program declares several host arrays and calls a PL/SQL stored procedure (`GET_EMPLOYEES` in the `CALLDEMO` package). The PL/SQL procedure returns up to `ASIZE` values. The program keeps calling `GET_EMPLOYEES`, getting `ASIZE` arrays each time, and printing the values, until all rows have been retrieved.

SAMPLE10
Connects to an Oracle database using your username and password and prompts for a SQL statement. You can enter any legal SQL statement, but you must use regular SQL syntax, not embedded SQL. Your statement is processed. If the statement is a query, the rows fetched are displayed (dynamic SQL Method 4).

SAMPLE11
Fetches from the `emp` table, using a cursor variable. The cursor is opened in the stored PL/SQL procedure `open_cur`, in the `EMP_DEMO_PKG` package.

SAMPLE12
Demonstrates how to do array fetches using dynamic SQL Method 4.

SCDEMO1
Demonstrates how the scrollable cursor can be used with Oracle dynamic SQL Method 4. Scrollable cursor can also be used with ANSI dynamic SQL Method 4.

SCDEMO2
Demonstrates the use of scrollable cursor with host arrays.

SQLVCP
Demonstrates how you can use the `sqlvcp()` function to determine the actual size of a `VARCHAR` struct. The size is then used as an offset to increment a pointer that steps through an array of `VARCHARs`.

This program also demonstrates how to use the `SQLStmtGetText()` function to retrieve the text of the last SQL statement that was executed.

WINSAM
Adds new employee records to the personnel database and checks database integrity. You can enter as many employee names as you want and perform the SQL commands by selecting the appropriate buttons in the *Employee Record* dialog box. This is a GUI version of the sample program.

Building the Demonstration Tables

To run the sample programs, you must have a database account with the username `scott` and the password `tiger`. Also, you must have a database with the sample tables `emp` and `dept`. This account is included in the starter database for your Oracle Database 10*g* server. If the account does not exist on your database, create the account before running the sample programs. If your database does not contain `emp` and `dept` tables, you can use the `demobld.sql` script to create them.

See Also: *Oracle Database Platform Guide for Windows*

To build the sample tables:

1. Start SQL*Plus
2. Connect as username `scott` with the password `tiger`.
3. Run the `demobld.sql` script:

```
SQL> @ORACLE_BASE\ORACLE_HOME\sqlplus\demo\demobld.sql;
```

Building the Sample Programs

You can build the sample programs in two ways:

- Using the `pcmake.bat` file provided
- Using Microsoft Visual C++ 6.0

Using pcmake.bat

The `pcmake.bat` file for compiling Pro*C/C++ demos is found in the following location:

```
ORACLE_BASE\ORACLE_HOME\precomp\demo\proc
```

This batch file is designed to illustrate how Pro*C/C++ applications can be built at the command prompt.

In order to use this batch file, Microsoft Visual Studio must be installed. The environment variable `MSVCDir` must be set. Pro*C/C++ command line options and linker options vary depending on your application.

You can use this file to build a demo, to build `sample1` for example:

1. Navigate to the location of the demo file and enter the following at the command prompt:

```
C:\> CD ORACLE_BASE\ORACLE_HOME\precomp\demo\proc\sample1
```

2. Enter the following:

```
% pcmake sample1
```

Using Microsoft Visual C++

Microsoft Visual C++ 6.0 project files have an extension of `.dsp`. The `.dsp` files in the *ORACLE_BASE*`\`*ORACLE_HOME*`\precomp\demo\proc` directory guide and control the steps necessary to precompile, compile, and link the sample programs.

Pro*C/C++, SQL*Plus, and the Object Type Translator have been integrated into the Microsoft Visual C++ sample project files. You do not have to run Pro*C/C++, SQL*Plus, and the Object Type Translator separately before compilation.

See Also:

- "Setting the Path for the Sample .pre Files" on page 3-7
- Appendix A, "Integrating Pro*C/C++ into Microsoft Visual C++"
- *Pro*C/C++ Programmer's Guide* for more information on Object Type Translator

To build a sample program:

1. Open a Visual C++ project file, such as `sample1.dsp`.
2. Check the paths in the project file to ensure that they correspond to the configuration of your system. If they do not, change the paths accordingly. Your system may produce error messages if the paths to all components are not correct.

> **Note:** All of the sample programs were created with `C:\oracle\ora92` as the default drive.

1. Select **Build > Rebuild All**. Visual C++ creates the executable.

Setting the Path for the Sample .pre Files

By default the sample `.pre` files search for their corresponding `.pc` files in the `C:\oracle\ora92` directory where `C:\` is the drive that you are using, and `oracle\ora92` represents the location of the Oracle home. If the Oracle base and Oracle home directories are different on your computer, you must change the directory path to the correct path.

To change the directory path for a sample .pre file:

1. In Pro*C/C++, open the `.pre` file.
2. Double-click the filename in the Input File area to display the Input File dialog box.
3. Change the directory path to the correct path.
4. Click **Open**.

A

Integrating Pro*C/C++ into Microsoft Visual C++

This appendix describes how to integrate Pro*C/C++ into the Microsoft Visual C++ integrated development environment.

This appendix contains these topics:

- Integrating Pro*C/C++ within Microsoft Visual C++ Projects
- Adding Pro*C/C++ to the Tools Menu

Integrating Pro*C/C++ within Microsoft Visual C++ Projects

This section describes how to fully integrate Pro*C/C++ within Microsoft Visual C++ projects.

All the precompiler errors and warnings are displayed in the output box where Microsoft Visual C++ displays compiler and linker messages. You do not have to precompile a file separately from the Microsoft Visual C++ build environment. More importantly, Microsoft Visual C++ maintains the dependencies between `.c` and `.pc` files. Microsoft Visual C++ maintains the dependency and precompile files, if needed.

All of the procedures in this section are performed within Microsoft Visual C++.

Specifying the Location of the Pro*C/C++ Executable

For Microsoft Visual C++ to run Pro*C/C++, it must know the location of the Pro*C/C++ executable. If Microsoft Visual C++ was installed before any Oracle release 9.2 products were installed, then you must add the directory path.

To specify the location of the Pro*C/C++ executable:

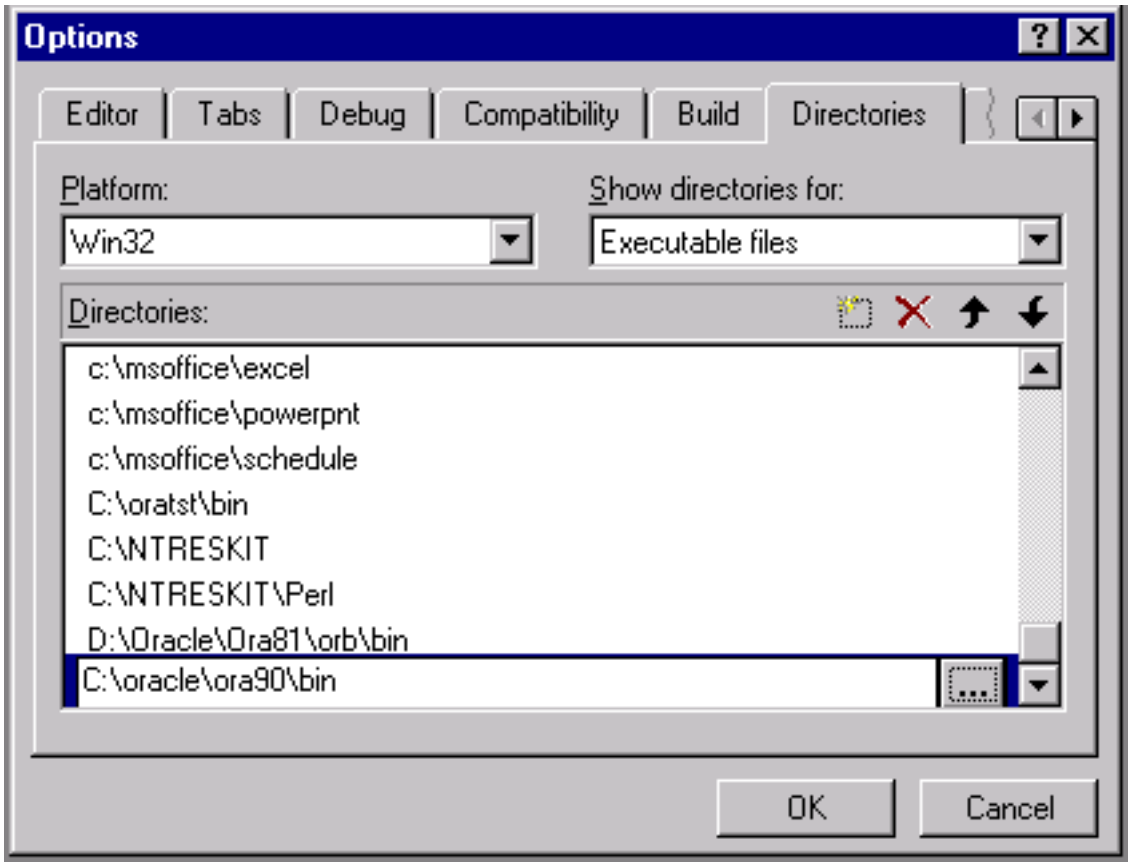
1. Select **Options** from the Tools menu.

   The Options dialog appears.

2. Click the **Directories** tab.
3. Select **Executable files** from Show directories **For**.
4. Scroll to the bottom of the Directories box and click the dotted rectangle.
5. Enter the *ORACLE_BASE*`\`*ORACLE_HOME*`\bin` directory. For example:

   ```
   C:\oracle\ora92\bin
   ```

6. Click **OK**.



Specifying the Location of the Pro*C/C++ Header Files

To specify the location of the Pro*C/C++ header files:

1. Select **Options** from the Tools menu. The Options dialog appears.
2. Click the **Directories** tab.
3. Select **Include Files** from the Show Directories For list.
4. Scroll to the bottom of the Directories box and click the **dotted rectangle**.
5. Enter the *ORACLE_BASE**ORACLE_HOME*\precomp\public directory. For example:

   ```
   C:\oracle\ora92\precomp\public
   ```

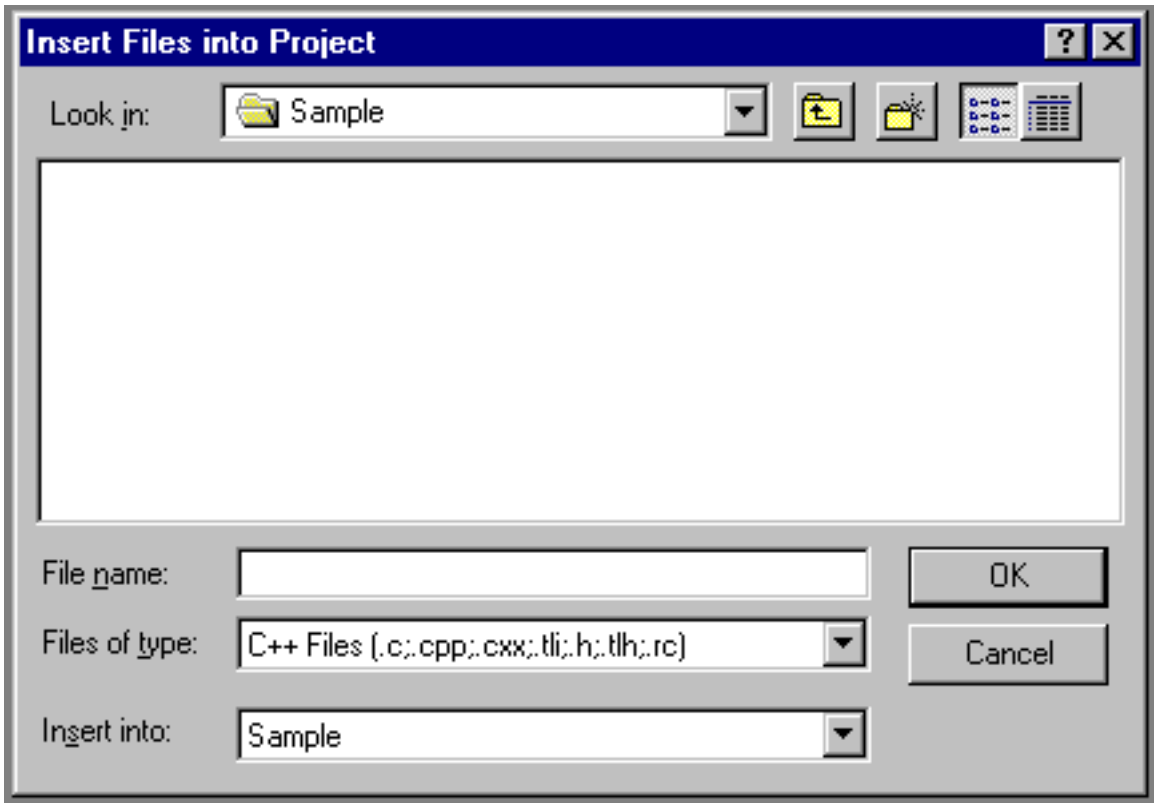
6. Click **OK**.

Adding .pc Files to a Project

After you create a project, you need to add the `.pc` files.

To add a `.pc` file to a project:

1. Select **Add To Project** from the Project menu and then select Files. The Insert Files into Project dialog appears.



2. Select **All Files** from the Files list.
3. Select the **.pc** file.
4. Click **OK**.

Adding References to .c Files to a Project

For each `.pc` file, you need to add a reference to the `.c` file that will result from precompiling.

To add a reference to a `.c` file to a project:

1. Select **Add To Project** from the Project menu, and then select Files. The Insert Files into Project dialog appears.
2. Type the name of the `.c` file in the File Name box.
3. Click **OK**. Because the `.c` file has not been created yet, Microsoft Visual C++ displays the following message: "The specified file does not exist. Do you want to add a reference to the project anyway?"
4. Click **Yes**.

Adding the Pro*C/C++ Library to a Project

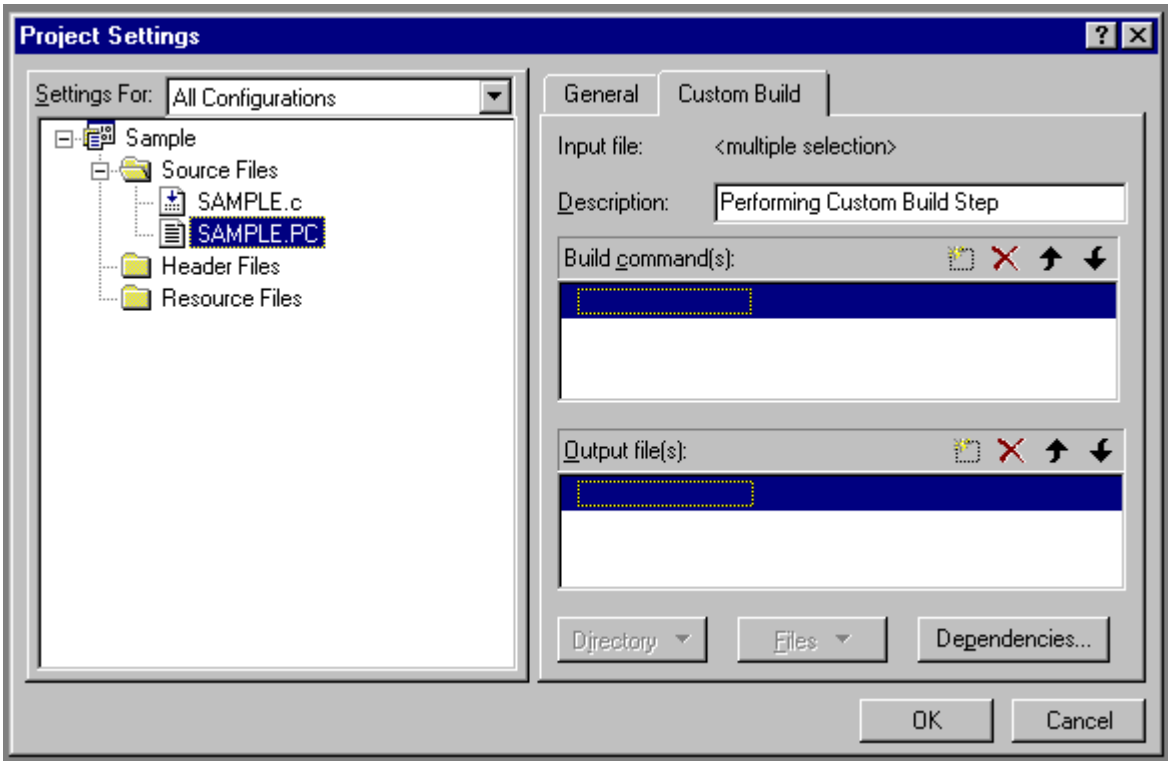
Pro*C/C++ applications must link with the library file `orasql10.lib`.

To add the Pro*C/C++ library to a project:

1. Select **Add To Project** from the Project menu, and then select **Files**. The Insert Files into Project dialog appears.

2. Select **All Files** from the Files list.
3. Select **orasql10.lib** from the *ORACLE_BASE**ORACLE_HOME*\precomp\lib\msvc directory.
4. Click **OK**.

Specifying Custom Build Options



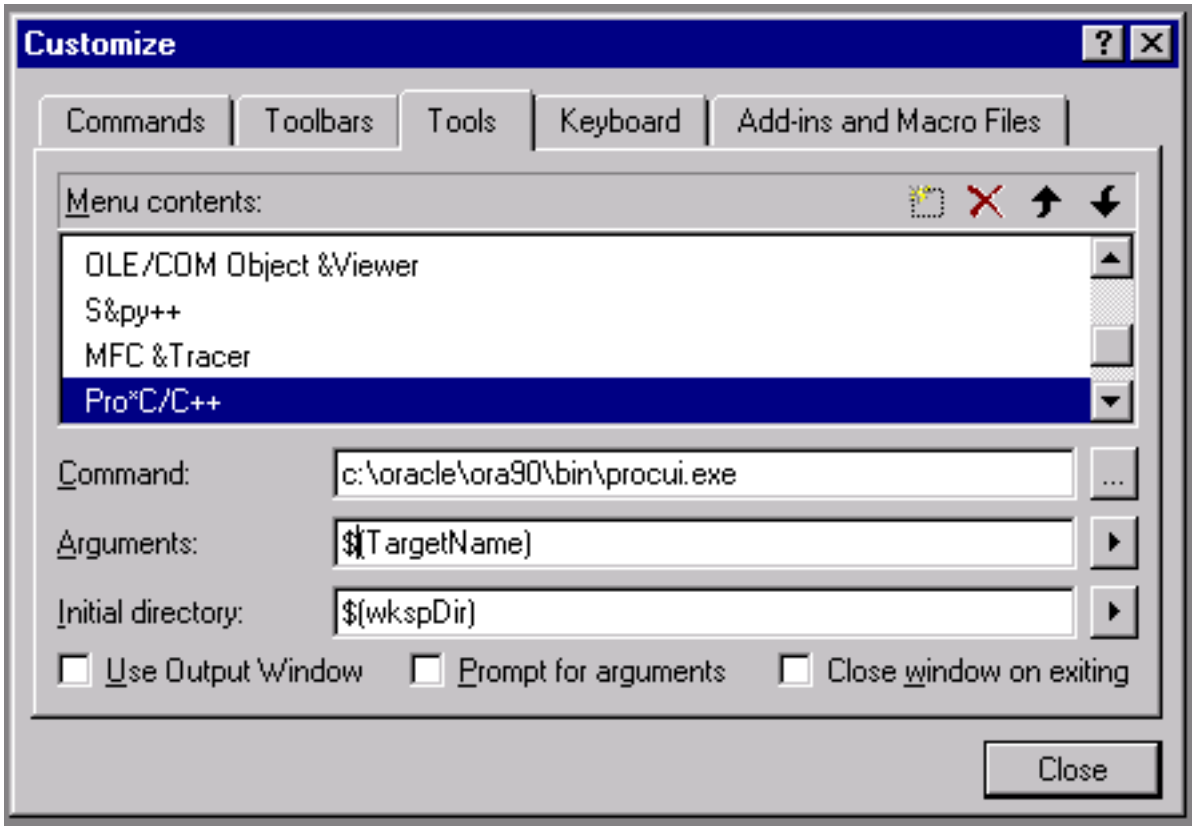
To specify Custom Build options:

1. In FileView, right-click a **.pc** file and select Settings. The Project Settings dialog appears with the Custom Build tab displayed.
2. In the Build commands box, on one line, set the build to use the same hardcoded path as that of the `$ORACLE_HOME` setting.
3. In the Output files box, enter one of the following:
   - If you are generating `.c` files, then enter `$(ProjDir)\$(InputName).c`.
   - If you are generating `.cpp` files, then enter `$(ProjDir)\$(InputName).cpp`.

   `$(ProjDir)` and `$MSDEVDIR` are macros for custom build commands in Microsoft Visual C++. When the project is built, Microsoft Visual C++ checks the date of the output files to determine whether they need to be rebuilt for any new modifications made to the source code.
4. Click **OK**.

See Also: Microsoft Visual C++ documentation

Adding Pro*C/C++ to the Tools Menu



You can include Pro*C/C++ as a choice in the Tools menu of Microsoft Visual C++.

To add Pro*C/C++ to the Tools menu:

1. From within Microsoft Visual C++, select **Customize** from the Tools menu. The Customize dialog appears.
2. Click the **Tools** tab.
3. Scroll to the bottom of the Menu contents box and click the dotted rectangle.
4. Enter the following text:

   ```
   Pro*C/C++
   ```

5. In the Command box, type the path and filename of the graphical Pro*C/C++ executable, or use the Browse button to the right of the box to select the file name. For example:

   ```
   C:\oracle\ora92\bin\procui.exe
   ```

6. In the Arguments box, enter the following text:

   ```
   $(TargetName)
   ```

   When you select **Pro*C/C++** from the Tools menu, Microsoft Visual C++ uses the `$(TargetName)` argument to pass the name of the current development project to Pro*C/C++. Pro*C/C++ then opens a precompile project with the same name as the opened project, but with a `.pre` extension in the project directory.

7. In the Initial directory box, enter the following text:

   ```
   $(WkspDir)
   ```

The Customize dialog should now look like the following graphic (although the Oracle home directory may be different on your computer).

8. Click **Close.** Microsoft Visual C++ adds Pro*C/C++ to the Tools menu.

Index

P

R

S

T