Oracle AppWizard for Microsoft Visual C++ User's Guide
Release 8.1.6
A73028-01

3
Understanding Your Application's Code

This chapter describes the building blocks that Oracle AppWizard for Microsoft Visual C++ uses to create an application. Using this information, you can easily customize your application.

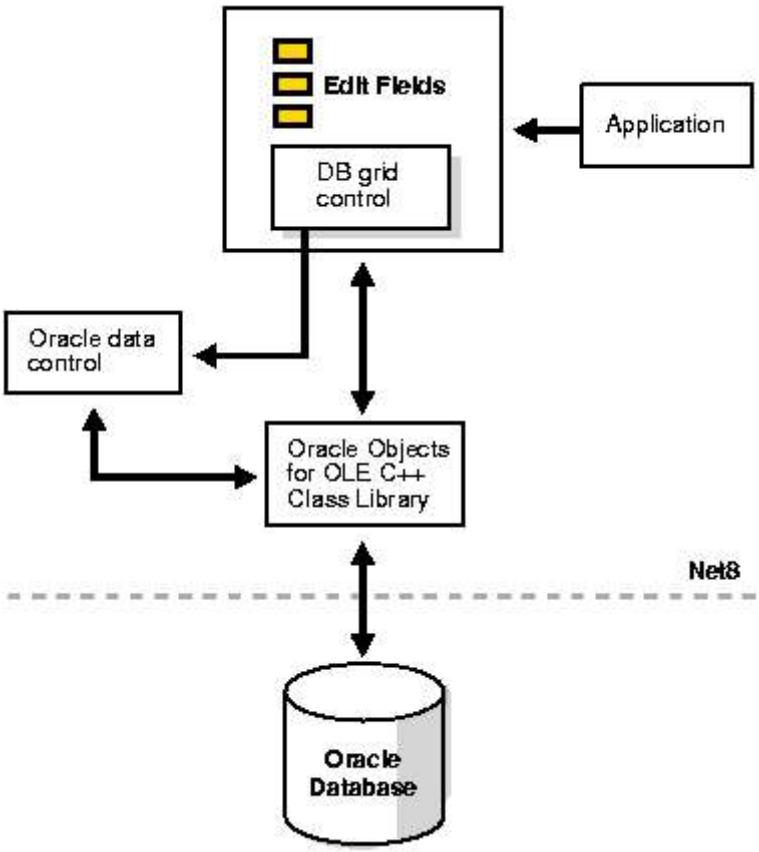
Specific topics discussed are:

Introduction

Oracle AppWizard for Microsoft Visual C++ uses Microsoft Foundation Classes (MFC) to provide the application framework. Oracle Objects for OLE C++ Class Library is used to create a starter application. Oracle AppWizard generates a complete set of source and resource files your Visual C++ project that you can customize as required for your business needs.

Figure 3-1, "Oracle AppWizard Framework" shows how the framework created by Oracle AppWizard uses the Oracle Objects for OLE C++ to interact with the Oracle database.

Figure 3-1 Oracle AppWizard Framework



Understanding the Generated Files

After you complete all Oracle AppWizard for Microsoft Visual C++ steps and have chosen the options appropriate for the application you want to create, Oracle AppWizard generates a complete set of files:

- Source and header
- Precompiled header
- Resource
- Miscellaneous files

Source and Header Files

Oracle AppWizard for Microsoft Visual C++ creates the following source and header files. Your files contain the actual name of your project, instead of *PRJNAME*.

File	Description
PRJNAME.h, *PRJNAME*.cpp	Derives from and implements the application class C*PRJNAME*App. This class provides member functions for initializing and running the application.
PRJNAME.clw	Used by the ClassWizard to store information about the classes in your project.
ConnDialog.h, ConnDialog.cpp	Derives from and implements the dialog class CConnDialog. This class enables the application to request connection information

File	Description
	from the user.
*PRJNAME*Doc.h, *PRJNAME*Doc.cpp	Derives from and implements the document class C*PRJNAME*Doc. This class also contains the application data, variables, and objects associated with the Oracle database. Data access occurs through Oracle Objects for OLE.
*PRJNAME*View.h, *PRJNAME*View.cpp	Derives from and implements the View class C*PRJNAME*View. This class displays document data graphically to the user and accepts and interprets user input as changes to the document.
*PRJNAME*Dynaset.h, *PRJNAME*Dynaset.cpp (for Single-Record and Multiple-Records Display Form)	Derives from and implements the Dynaset class *CPRJNAME*Dynaset. This class creates, manages, and accesses records in the database.
*PRJNAME*DynasetMaster.h, *PRJNAME*DynasetMaster.cpp (for Master-Detail Display Form)	Derives from and implements the Dynaset class C*PRJNAME*DynasetMaster. This class creates, manages, and accesses records in the database.
*PRJNAME*DynasetDetail.h, *PRJNAME*DynasetDetail.cpp (for Master-Detail Display Form)	Derives from and implements the Dynaset class C*PRJNAME*DDynasetDetail. This class creates, manages, and accesses records in the database.
ChildFrm.h, ChildFrm.cpp (for Multiple-Document interface application)	Derives from and implements the Child Window class CChildFrame. This class is used for MDI document frames.
oradc.h, oradc.cpp (for Multiple-Records and Master-Detail Display form)	Derives from and implements the Window class CORADC. This class is the wrapper class for Oracle Data Control, an ActiveX control and is used to access data for an application using multiple-record display mode or master-detail display mode.
*PRJNAME*Util.h, *PRJNAME*Util.cpp	Declare and implement stand-alone functions used by other classes. ProcessOO4OError() and DDX_FieldText() functions are defined in these files.
MainFrm.h, MainFrm.cpp	Derives from and implements the Frame class CMainFrame. This class provides the functionality of an overlapped single-document interface (SDI) or a pop-up window, along with members for managing the window.

Precompiled Header Files

Oracle AppWizard for Microsoft Visual C++ creates the following standard files.

StdAfx.h, StdAfx.cpp

These files are used to build a precompiled header file *PRJNAME*.PCH and a precompiled types file StdAfx.OBJ.

Resource Files

Oracle AppWizard creates the following standard header file and a main resource file:

Files	Description
Resource.h, *PRJNAME*.rc	These files contain the default menu definition, accelerator, and string tables for the generated application.
PRJNAME.rc2	

Files	Description
	This file is useful for including resources used by several different projects. Instead of having to create the same resources several times for different projects, you can put them in an RC2 file and include the RC2 file in the main RC file.
.\RES*PRJNAME*.ico	This is the icon file for the application. This icon appears when the application is minimized and is also used in the About box.
.\RES*PRJNAME*Doc.ico	This is the icon file for the child window in the Multiple Document Interface application.
.\RES\TOOLBAR.BMP	This bitmap file is used to represent your program or control in a toolbar or palette.

Miscellaneous Files

Oracle AppWizard for Microsoft Visual C++ also creates the following file.

ReadMe.txt

This file contains information about the files that Oracle AppWizard for Microsoft Visual C++ creates for your application.

Understanding the Code Within Generated Files

This section is divided into the following subsections, which explain the function of the code generated by Oracle AppWizard for Microsoft Visual C++:

This Section...	Describes the...
Oracle Objects for OLE classes	Building blocks for all aspects of application functionality
What Happens When the Application Starts	Code at the heart of application initialization, execution of SQL statements, connection to the database, display of database table columns
Navigational Flow	Code at the heart of record navigation
Data Manipulation Flow	Code at the heart of record insertion, update, and deletion
Generated Code for a Multiple-Record Display Form	Code used to create an application whose forms display multiple-record at one time

Note:

The generated code described in all but the last list item above is based on an application using single-record display mode exclusively.

Oracle Objects for OLE classes

Oracle AppWizard generates Oracle Object for OLE C++ code to provide connectivity and data access to the Oracle database. The following table illustrates the classes that Oracle AppWizard uses to communicate with the Oracle database, using the Oracle Objects for OLE C++ Class Library.

Class	Description

Class	Description
ODatabase	A database object representing an Oracle database
ODynaset	Creates, manages, and accesses records in the Oracle database
OField	A single column of data in an Oracle database record
OValue	Stores values of varying types in the Oracle database

The oracl.h header file in ORACLE_HOME\OO4O\CPP\INCLUDE contains the declarations for Oracle Objects for OLE classes.

What Happens When the Application Starts

This subsection describes the methods that control the following activities either during or after start-up as you work through the sequence of windows designed to build the starter application.

- Initialization of the Oracle Objects for OLE C++ Class Library in the generated application
- Connection to database
- Execution of SQL statements
- Table column displays

Initializing Oracle Object for OLE C++ Class Library

When the application starts, the OStartup() call in C*PRJNAME*App::InitInstance() initializes the Oracle Object for OLE C++ Class Library as described below:

Note:

Text in **bold** type represents Oracle code.

```
BOOL CTestApp::InitInstance()
{


    ...
// initializing Oracle Object for OLE C++ Class Library
OStartup();
   ...


}
```

Connecting to Database

The application framework creates a document object (C*PRJNAME*Doc class) that stores the application data. During document initialization, it calls DbConnect() to initialize the database (m_database) associated with the document and connects to it, using the following declaration:

```
class CPRJNAMEDoc : public CDocument
{
```

```
  ...

    // Attributes
    public:



      ODatabase m_database;
      CPRJNAMEDynaset m_PRJNAMEDynaset;



    // Implementation
    public:



virtual ~CTestDoc();
bool DbConnect();


...
};
```

Constructor method of C*PRJNAME*Doc class:

```
CPRJNAMEDoc::CPRJNAMEDoc()
{


while (!m_database.IsOpen())



    {
     if (DbConnect() == false)



return;



}

}
```

To initialize and connect to the database, the application:

- Creates a dialog box (connDlg) to request database connection information
- Connects to the database
- If the connection fails, the application processes the error

The method C*PRJNAME*Doc::DbConnect() implements the database connection as shown below:

```
bool CPRJNAMEDoc::DbConnect()
{
```

```
   CConnDialog connDlg;
   oresult dbresult;
   int dlgResult;


// a dialog box to get connection information from users


 dlgResult = connDlg.DoModal();
 if (dlgResult == IDOK)


 {
// Connect to the database


 dbresult = m_database.Open(connDlg.GetDbAlias(),
 connDlg.GetUsername(), connDlg.GetPassword());
 if (dbresult == OFAILURE)
 {


// processing error message


  ProcessOO4OError(&m_database);
  }
  return(true);


}
else


 return(false);


}
```

Executing SQL Statements

If successfully connected, the framework creates a view (C*PRJNAME*View class) for the application, using the following declaration:

```
class CPRJNAMEView : public CFormView
{


 ...
 public:



CTestDynaset *m_pDynaset;
bool m_bEditingRecord;  //if the record must be edited



// Operations
protected:
```

```
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
virtual void OnInitialUpdate(); // called first time after construct




...


// Implementation
public:


 virtual ~CPRJNAMEView();


protected:
...
 void PerformMove(int nCommand);
// Generated message map functions
protected:


...
afx_msg void OnMoveNext();
afx_msg void OnMovePrev();
afx_msg void OnMoveFirst();
afx_msg void OnMoveLast();
afx_msg void OnUpdateMoveNext(CCmdUI* pCmdUI);
afx_msg void OnUpdateMovePrev(CCmdUI* pCmdUI);
afx_msg void OnUpdateMoveFirst(CCmdUI* pCmdUI);
afx_msg void OnUpdateMoveLast(CCmdUI* pCmdUI);
afx_msg void OnAddNewRecord();
afx_msg void OnUpdateAddNewRecord(CCmdUI* pCmdUI);
afx_msg void OnDeleteRecord();
afx_msg void OnUpdateDeleteRecord(CCmdUI* pCmdUI);
afx_msg void OnUpdateRecord();
afx_msg void OnUpdateUpdateRecord(CCmdUI* pCmdUI);
afx_msg void OnCancelRecord();
afx_msg void OnUpdateCancelRecord(CCmdUI* pCmdUI);
afx_msg void OnChangeEdit();
...


}
```

The constructor method for the C*PRJNAME*View class initializes its member variables as:

```
CPRJNAMEView::CPRJNAMEView()
: CFormView(CPRJNAMEView::IDD)
{
...
m_pDynaset = NULL;
m_bEditingRecord = false;
...
 }
```

Before the view appears, the application framework calls the C*PRJNAME*View::OnInitialUpdate() method to perform initialization, requiring the following document information:

- A view needs to display information from a table. To get information from the document for this purpose, the m_pDynaset member variable in the C*PRJNAME*View class points to the m_*PRJNAME*Dynaset

member variable in the C*PRJNAME*Doc class.
- A dynaset needs to be associated with a set of records in a table. To open a dynaset in a database associated with the document, the application calls ODynaset::OpenQuery() method.

This implements the *CPRJNAME*View::OnInitialUpdate() method, illustrated below.

```
void CPRJNAMEView::OnInitialUpdate()
{


...
m_pDynaset = &GetDocument()->m_PRJNAMEDynaset;
m_pDynaset->OpenQuery(GetDocument()->m_database);
...


}
```

The C*PRJNAME*Dynaset class represents a set of records in a table. It creates, manages, and accesses records in the database, using the declaration shown below. In this declaration, data member variables have comments that indicate which real columns they represent.

```
class CTestDynaset : public ODynaset
{
...
public:


 // strings needed for creating the queries
 CString m_strSQLQuery;  // the query to be sent to the database
 CString m_strSQLSelect; // the select portion
 CString m_strSQLFilter; // the where portion
 CString m_strSQLSort;   // the order by portion
// Field/Param Data
OField m_Column1; //for COL1
OField m_Column2; //for COL2


...
// Operations
public:


void OpenQuery(ODatabase theDB);
void CreateSQLSelect();
void CreateSQLFilter();
void AddFilter(CString strFilter);
void CreateSQLSort();
void ResetToDefaultFilter();
void RefreshQuery();


...
}
```

To open a dynaset associated with the document, the method *CPRJNAME*Dynaset::OpenQuery() is called in *CPRJNAME*View::OnInitialUpdate(). The *CPRJNAME*Dynaset::OpenQuery() method takes the following actions:

- Creates a query statement to be executed
- Asks the database for a set of records and sets up a dynaset to access them

- Binds data controls with the OField member variables (m_Column1, m_Column2,...etc.) in the CPRJNAMEDynaset class

The *CPRJNAME*Dynaset::OpenQuery() method is implemented as illustrated below:

```
 // opens the query
 void CPRJNAMEDynaset::OpenQuery(ODatabase theDB)
{


 oresult dbresult;


 // m_strSQLQuery, m_strSQLSelect, m_strSQLFilter are CString objects.
 // These are member variables in CPRJNAMEDynaset class
 // create a query statement to be executed by the Open() call
 m_strSQLQuery = m_strSQLSelect;
 if (m_strSQLFilter)
   m_strSQLQuery += m_strSQLFilter;
 if (m_strSQLSort)
  m_strSQLQuery += m_strSQLSort;
// executing a SQL statement
dbresult = Open(theDB, m_strSQLQuery);
 // binding edit controls with columns in the table
  m_Column1 = GetField("COL1");
  m_Column2 = GetField("COL2");


 ...


}
```

Displaying Columns in a Table

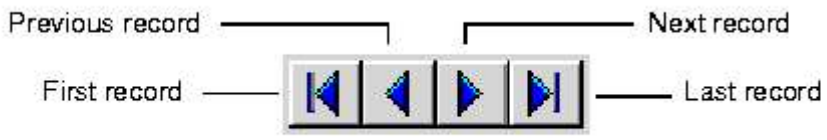
A view displays the stored data in the associated document. Data in both the document and the view must be consistent. Therefore, the OField member variables (the columns in a database table) of the *CPRJNAME*Dynaset class and the edit controls in the view must be validated and exchanged, using the *CPRJNAME*View::DoDataExchange() method, which is implemented as:

```
void CPRJNAMEView::DoDataExchange(CDataExchange* pDX)
{


...
 // performing data exchange
 DDX_FieldText(pDX, IDC_COL1,
m_pDynaset->m_Column1, m_pDynaset);
 DDX_FieldText(pDX, IDC_COL2,
m_pDynaset->m_Column2, m_pDynaset);


    ...
}
```

Navigational Flow

This section explains the code generated by Oracle AppWizard for Microsoft Visual C++ to allow navigation through a dynaset record set.

These four controls enable your application users to navigate through records:



These navigational controls work as shown in the following two tables. Event handlers handle the COMMAND message, and Message handlers handle UPDATE_COMMAND_UI messages.

Action	Button Control ID	Event Handler	Description
Move to first record	ID_RECORD_MOVEFIRST	OnMoveFirst	Move to first record in the dynaset's result set.
Move to previous record	ID_RECORD_MOVEPREV	OnMovePrev	Move to previous record in the dynaset's result set.
Move to next record	ID_RECORD_MOVENEXT	OnMoveNext	Move to next record in the dynaset's result set.
Move to last record	ID_RECORD_MOVELAST	OnMoveLast	Move to last record in the dynaset's result set.

Action	Button Control ID	Message Handler for Update Command	Description
Move to first record	ID_RECORD_MOVEFIRST	OnUpdateMoveFirst	Update first record in the dynaset's result set.
Move to previous record	ID_RECORD_MOVEPREV	OnUpdateMovePrev	Update previous record in the dynaset's result set.
Move to next record	ID_RECORD_MOVENEXT	OnUpdateMoveNext	Update next record in the dynaset's result set.
Move to last record	ID_RECORD_MOVELAST	OnUpdateMoveLast	Update last record in the dynaset's result set.

The navigational operations all use the CPRJNAMEView::PerformMove() method to perform the requested navigation. To explain the generated code for the navigational operations, we must explain the generated code for the CPRJNAMEView::PerformMove() method.

How Record Navigation Works

All navigational operations use the *CPRJNAME*View::PerformMove() method.

After a request for record navigation by the user and before any navigation occurs, the application verifies whether or not the current record must be updated or inserted into the database by calling *CPRJNAME*View::PerformMove(). The method performs the following operations:

- Checks if the record can be edited
- Performs data exchange between the view and the dynaset

- If the record has been modified, updates the record in the database
- Performs the requested navigation

*CPRJNAME*View::PerformMove() method is implemented as:

```
void CPRJNAMEView::PerformMove(int nCommand)
{


// nCommand is the control ID of the navigational button and represents
// the requested operation.


 oresult dbresult;


// Verify whether or not the record can be edited.
if (m_pDynaset->GetEditMode() != ODYNASET_EDIT_NEWRECORD)



m_pDynaset->StartEdit();



// exchange data between the the view and the dynaset
if (!UpdateData())



return;



// if the record changes, update the record in the dynaset
if (m_pDynaset->get_CurrentRowModified())
{
dbresult = m_pDynaset->Update();
if (dbresult == OFAILURE)



{

 ProcessOO4OError(m_pDynaset);
 return;

}

}
else


dbresult = m_pDynaset->CancelEdit();


switch(nCommand) // perform action, depends on the requeset navigation
{


  case ID_RECORD_MOVENEXT:
```

```
m_pDynaset->MoveNext();
break;


 case ID_RECORD_MOVEPREV:


m_pDynaset->MovePrev();
break;


 case ID_RECORD_MOVEFIRST:


m_pDynaset->MoveFirst();
break;


 case ID_RECORD_MOVELAST:


m_pDynaset->MoveLast();
break;


}
// exchange data between the view and the dynaset
UpdateData(FALSE);
m_bEditingRecord = false;
}
```

Navigating to the First Record in the Database

When the current record is not the first record and the user clicks the "First Record" button, corresponding to control ID = ID_RECORD_MOVEFIRST, users navigate to the first record.

The *CPRJNAME*View::OnUpdateMoveFirst() method enables/disables the "First Record" button and is implemented as:

```
Void CTestView::OnUpdateMoveFirst(CCmdUI* pCmdUI)
{


pCmdUI->Enable((m_pDynaset->IsOpen() == TRUE



&& m_pDynaset->IsFirst() ==    FALSE) ? true : false);
```

```
}
```

When the button with control ID = ID_RECORD_MOVEFIRST is clicked, CPRJNAMEView::OnMoveFirst() method is called to handle the "Move to First Record" event.

CPRJNAMEView::OnMoveFirst() method is implemented as:

```
// move to the first record
void CPRJNAMEView::OnMoveFirst()
{


PerformMove(ID_RECORD_MOVEFIRST);


}
```

Navigating to the Last Record

When the current record is not the last record and the user clicks the "Last Record" button, corresponding to control ID = ID_RECORD_MOVELAST, users navigate to the last record.

The CPRJNAMEView::OnUpdateMoveLast() method enables/disables the "Move to Last Record" button as shown below:

```
void CTestView::OnUpdateMoveLast(CCmdUI* pCmdUI)
{


pCmdUI->Enable((m_pDynaset->IsOpen() == TRUE



&& m_pDynaset->IsLast() == FALSE) ? true : false);


}
```

When the button with control ID = ID_RECORD_MOVELAST is clicked, CPRJNAMEView::OnMoveLast() method is called to handle the "Move to Last Record" event.

This is the implementation of CPRJNAMEView::OnMoveLast() method:

```
// move to the last record
void CPRJNAMEView::OnMoveLast()
{


PerformMove(ID_RECORD_MOVELAST);


}
```

Navigating to the Previous Record

When the current record is not the first record and the user clicks the "Previous Record" button, corresponding to control ID = ID_RECORD_MOVEPREV, users navigate to the previous record.

The CPRJNAMEView::OnUpdateMovePrev() method enables/disables the "Previous Record" button as shown below:

```
void CTestView::OnUpdateMovePrev(CCmdUI* pCmdUI)

{


        CTestView::OnUpdateMoveFirst(pCmdUI);


}
```

Clicking the "Previous Record" button calls the CPRJNAMEView::OnMovePrev() method to handle the "Move to Previous Record" event. It is implemented as shown below:

```
// move to the previous record
void CPRJNAMEView::OnMovePrev()
{


PerformMove(ID_RECORD_MOVEPREV);


}
```

Navigating to the Next Record

When the current record is not the last record and the user clicks the "Next Record" button, corresponding to control ID = ID_RECORD_MOVENEXT, users navigate to the next record.

The CPRJNAMEView::OnUpdateMoveNext() method enables/disables the "Next Record" button as shown below:

```
void CTestView::OnUpdateMoveNext(CCmdUI* pCmdUI)
{


CTestView::OnUpdateMoveLast(pCmdUI);


}
```

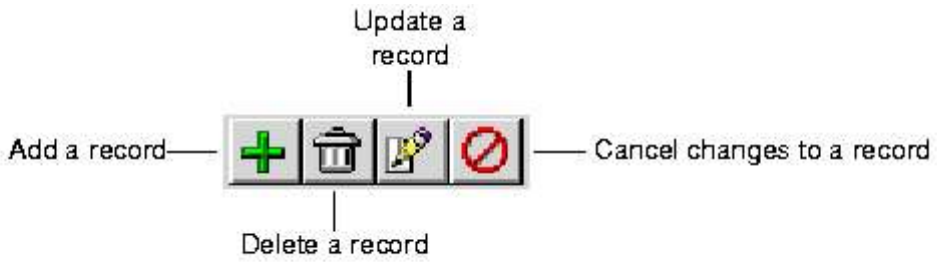
Clicking the "Next Record" button calls the CPRJNAMEView::OnMoveNext() method to handle the "Move to Next Record" event. It is implemented as shown below:

```
// move to the next record
void CPRJNAMEView::OnMoveNext()
{


PerformMove(ID_RECORD_MOVENEXT);


}
```

Data Manipulation Flow

This section explains the code that Oracle AppWizard for Microsoft Visual C++ generates to manipulation data in a table.

The following four controls enable application users to add, delete, and update records, or to cancel record updates before being sent to the database:



The controls illustrated above correspond to the ODynaset class methods shown in the table below. Event handlers handle the COMMAND message, and Message handlers handle UPDATE_COMMAND_UI messages.

Action	Button Control ID	Event Handler	Description
Add a new record	ID_RECORD_ADDNEW	OnAddNewRecord	Add a new record to the dynaset's result set
Delete a record	ID_RECORD_DELETE	OnUpdateRecord	Delete a record from the dynaset's result set
Update a record	ID_RECORD_UPDATE	OnDeleteRecord	Commit the changes for the current record
Cancel changes to a record	ID_RECORD_CANCEL	OnCancelRecord	Cancels the changes to the current record

Action	Button Control ID	Message Handler
Add new record	ID_RECORD_ADDNEW	OnUpdateAddNewRecord
Delete a record	ID_RECORD_DELETE	OnUpdateUpdateRecord
Update a record	ID_RECORD_UPDATE	OnUpdateDeleteRecord
Cancel changes to a record	ID_RECORD_CANCEL	OnUpdateCancelRecord

Adding a New Record

When the current record is not being updated and the user clicks the "Add New Record" button, corresponding to control ID = ID_RECORD_ADDNEW, users add a new record to the database.

The CPRJNAMEView::OnUpdateAddNewRecord() method enables/disables the "Add New Record" button as shown below:

```
void CTestView::OnUpdateAddNewRecord(CCmdUI* pCmdUI)
{

CmdUI->Enable(m_pDynaset->IsOpen() == TRUE && !m_bEditingRecord);

}
```

Clicking the "Add New Record" button, calls the CPRJNAMEView::OnAddNewRecord() method to handle the "Add Record" event as shown below:

```
// Allows the user to add a new record to the table
void CPRJNAMEView::OnAddNewRecord()
{


if (m_pDynaset->AddNewRecord() == OSUCCESS)
{



UpdateData(FALSE);
m_bEditingRecord = true;




}
else
{



ProcessOO4OError(m_pDynaset);




}


}
```

Updating a Record

When the current record is not being updated and the user clicks the "Update Record" button, corresponding to control ID = ID_RECORD_UPDATE, users update a record in the database.

The CPRJNAMEView::OnUpdateUpdateRecord() method enables/disables the "Update Record" button as shown below:

```
void CTestView::OnUpdateUpdateRecord(CCmdUI* pCmdUI)
{


pCmdUI->Enable(m_pDynaset->IsOpen() == TRUE && m_bEditingRecord);


}
```

Clicking the "Update Record" button calls the CPRJNAMEView::OnUpdateRecord() method to handle the "Update Record" event as shown below:

```
// Updates the changes the user has made to the current record of the table
void CCPRJNAMEView::OnUpdateRecord()
{


oresult dbresult;
if (m_pDynaset->GetEditMode() != ODYNASET_EDIT_NEWRECORD)
```

```
 m_pDynaset->StartEdit();



if (!UpdateData())



 return;



if (m_pDynaset->get_CurrentRowModified())
{
dbresult = m_pDynaset->Update();
if (dbresult == OFAILURE)
    {



 ProcessOO4OError(m_pDynaset);
 return;
    }


  }


  else
  dbresult = m_pDynaset->CancelEdit();
  UpdateData(FALSE);
  m_bEditingRecord = false;


}
```

Deleting a Record

When the current record is not being updated and the user clicks the "Delete Record" button, corresponding to control ID = ID_RECORD_DELETE, users delete a record in the database.

The CPRJNAMEView::OnUpdateDeleteRecord() method enables/disables the "Delete Record" button as shown below:

```
void CTestView::OnUpdateDeleteRecord(CCmdUI* pCmdUI)
{


pCmdUI->Enable(m_pDynaset->IsOpen() == TRUE && !m_bEditingRecord);


}
```

Clicking the "Delete Record" button calls the CPRJNAMEView::OnDeleteRecord() method to handle the "Delete Record" event as shown below:

```
// Deletes the current record from the table
void CCPRJNAMEView::OnDeleteRecord()
{
```

```
bool bWasLast = m_pDynaset->IsLast() == TRUE ? true : false;
if (m_pDynaset->DeleteRecord() == OSUCCESS)
{



 if (!bWasLast)



m_pDynaset->MoveNext();



else



m_pDynaset->MovePrev();



UpdateData(FALSE);
m_bEditingRecord = false;



}
else
{



ProcessOO4OError(m_pDynaset);



}

}
```

Cancelling Changes to a Record

When the current record is being updated and the user clicks the "Cancel record changes" button, corresponding to control ID = ID_RECORD_CANCEL, users cancel changes they have entered, but have not yet committed, to a record in the database.

The CPRJNAMEView::OnUpdateCancelRecord() method enables/disables the "Cancel Changes" button as shown below:

```
void CTestView::OnUpdateCancelRecord(CCmdUI* pCmdUI)
{


pCmdUI->Enable(m_pDynaset->IsOpen() == TRUE && m_bEditingRecord);
```

```
}
```

Clicking the "Cancel Changes" button, calls the CPRJNAMEView::OnCancelRecord() method to handle the "Cancel the change" event as shown below:

```
// Cancels the changes to the current record.
void CCPRJNAMEView::OnCancelRecord()
{


if (m_pDynaset->GetEditMode() != ODYNASET_EDIT_NOEDIT)
if (m_pDynaset->CancelEdit() == OFAILURE)



ProcessOO4OError(m_pDynaset);



UpdateData(FALSE);
m_bEditingRecord = false;


}
```

Generated Code for a Multiple-Record Display Form

This subsection explains the generated code for applications using multiple-record display forms. The previous discussion in this section dealt with generated code for single-record display forms.

Note:

Text in **bold** type represents Oracle code.

Oracle AppWizard for Microsoft Visual C++ uses Oracle Data Control class CORADC to display and access records in this mode. The CPRJNAMEView class is defined as follows:

```
class CPRJNAMEView : public CFormView
{


...
public:



CTestDynaset *m_pDynaset;
CORADC m_dataControl // Used by Oracle control to display
                  and access data



...
```

```
}
```

Data exchange takes place, using the Oracle Data Control Object (m_dataControl) in the CPRJNAMEView::DoDataExchange() as follows:

```
void CPRJNAMEView::DoDataExchange(CDataExchange* pDX)
{


// performing data exchange
// IDC_ORADC is the control ID for the Oracle Data Control grid
DDX_Control(pDX, IDC_ORADC, m_dataControl);


}
```