Лабораторная работа № 15
Добавление функциональности в класс просмотра

Создание пункта меню Work(работа)

Мастер приложения создал для нас полноценный текстовой редактор, на основе которого и будет осуществляться дальнейшая работа. Функции редактора нам не нужны, их можно использовать для более простого вывода текста на экран. Но, что самое главное, в проекте уже есть заготовки классов для работы нашей теплицы. Их только необходимо наполнить содержанием.

Пункт создан для того, чтобы дать команду программе начать обрабатывать план выращивания, то есть при запуске программы она ждет от оператора команды, когда необходимо начать отслеживание состояния среды.

Такой тактический ход вполне логичен. Ведь если бы мы создавали программу для работы реальной теплицы, то оставили бы возможность создания и редактирования планов выращивания различных растений и конфигурирования устройств теплицы, например указания их расположения и количества. И только после этого можно было бы запустить план выращивания.

Этот пункт меню был создан следующим образом: добавили новый пункт в ресурс Menu и присвоили ему ID=ID_WORK_STARTPLAN, как показано на рис. 112.

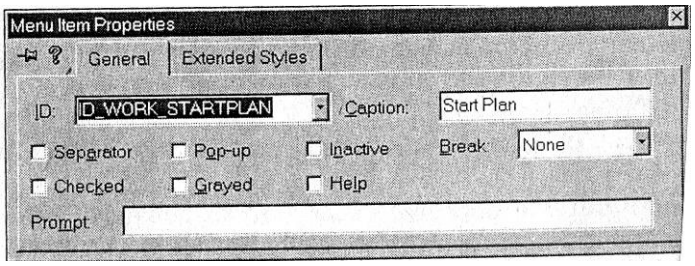

Рис.112. Добавление пункта меню в ресурс IDR_MAIN_FRAME

Ассоциация операции с пунктом меню

Для того чтобы этот пункт меню начал работать, его необходимо ассоциировать с операцией класса, для чего есть два пути:

1. Создать операцию при помощи Visual Studio ClassWizard (рис.113).
2. Создать обработку данного пункта в Rational Rose.

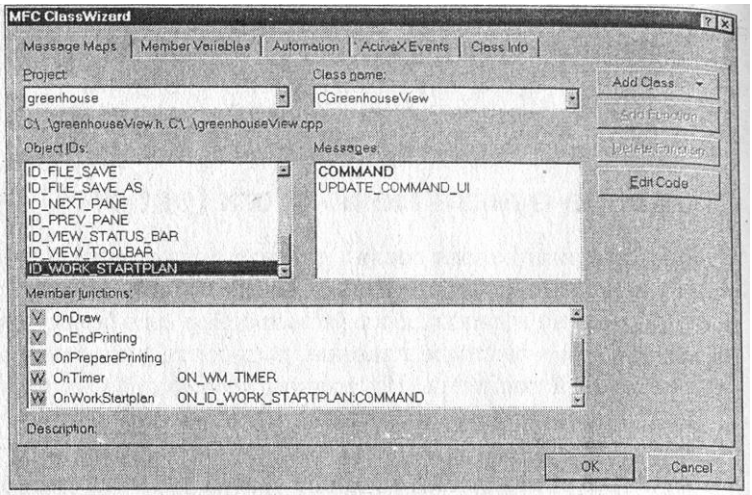

Рис.113. Создание обработчика пункта меню в ClassWizard

Второй вариант наиболее интересен для нас. Lля того чтобы им воспользоваться, перейдем в Rational Rose и проведем обновление модели из кода (Menu=>Tools=>Visual C++=>Update Model From Code).

Теперь можно создать его обработку в классе CGreenhouseView. Для этого необходимо выделить класс, например, в окне Browse или LogicalView и вызвать для него Model Assistant из контекстного меню.

Здесь как раз и проявляется глубина возможностей интеграции Rational Rose и VC++. После переключения во вкладку MFC мы получим доступ ко всем атрибутам и операциям текущего класса и всех родительских классов, для чего нет необходимости изучать иерархию классов MFC.

Для того чтобы создать обработчик. для команды ID_WORK_STARTPLAN, выделим пункт Command Handlers (обработчик команд) и из контекстного меню выберем пункт New Command Handler (новый обработчик). После чего активизируется окно, показанное на рис. 114.

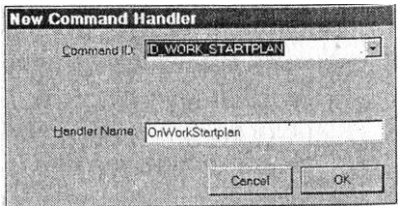

Рис.114. Создание обработчика команд меню в Rational Rose

Теперь в строке Command Handler образовался новый пункт OnWorkStartplan, который включает в себя информацию о новом обработчике (рис. 115).

Таким же образом сюда можно добавлять любые новые обработчики меню, причем их можно создать сначала здесь, а потом уже внести в ресурсы меню или другие ресурсы. Теперь можно добавить и другие операции, которые нам понадобятся для работы.

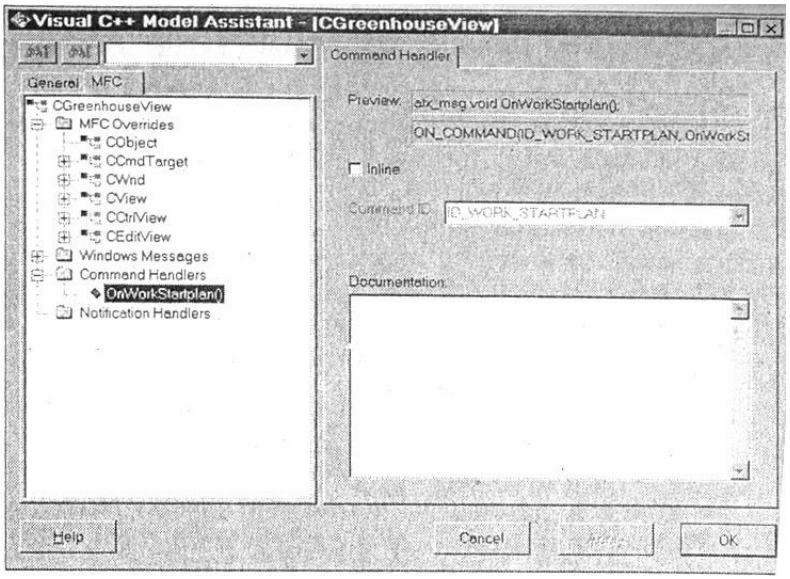

Рис. 115. Информация об обработчике команд в Model Assistant

Добавление операции в Model Assistant

Для вывода текущего состояния устройств и показателей датчиков необходимо создание операции, которая бы выводила эту информацию в главное окно программы. Назовем ее Message. Для того, чтобы добавить операцию, необходимо на строке Operations (операции) из контекстного меню выбрать пункт New Operation (новая операция) и заполнить параметры операции так, как показано на рис. 116. Не забудьте, что при заполнении имен и типов переменных сначала необходимо указать имя, а затем после

двоеточия, - ее тип. В любом случае всегда можно переименовать переменную, если что-то было заполнено неправильно при помощи RClick=>Rename.

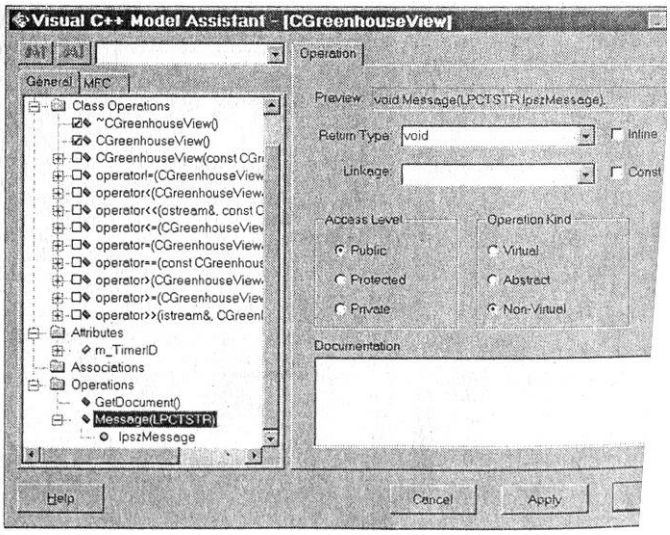

Рис.116. Ввод операции Message

Отслеживание сообщений таймера

Также нам понадобится отслеживать сообщения таймера. Ранее, до того как стала окончательно известна структура приложения, мы планировали встроить таймер непосредственно в класс EnvironmentalController, хотя и не отметали возможности определения таймера в каком-то внешнем классе. Теперь можно точно определить, куда включить данный класс. Включим инициализацию и обработку таймера в класс CGrennhouseView.

Для создания обработчика таймера необходимо во вкладке MFC выбрать пункт Windows Message (сообщения Windows) и поставить пункте OnTimer (рис. 117).

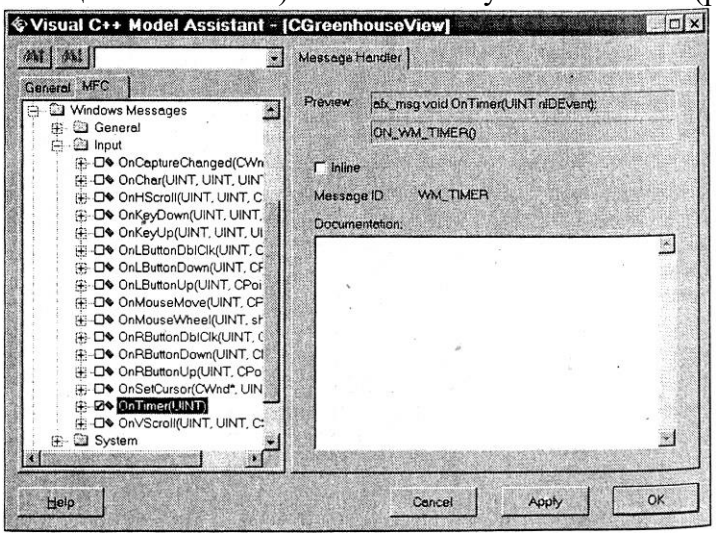

Рис.117. Включение обработчика таймера

Для нормальной работы таймера необходимо добавить в класс атрибут m_TimerID с типом UINT. Эта переменная будет хранить идентификатор таймера. Если данный идентификатор равен нулю, то таймер не активизирован, а если не ноль, то таймер работает. Эту информацию мы используем, когда будем заполнять обработчик OnWorkStartplan.

В случае когда приложение будет завершено до остановки таймера, чтобы высвободить этот ресурс Windows (ведь количество ресурсов ограничено), необходимо

при закрытии окна остановить таймер. Для этого поставьте отметку напротив операции Destroy Window путем Model Assistant=>MFC=>CWnd=>Destroy Windows=ON.

И, конечно же, для инициализации переменной таймера необходимо установить отметку для генерации конструктора класса.

Редактирование шаблона класса CGrennhouseView.

Таким образом, шаблон для создания класса CGrennhouseView полностью создан. Далее необходимо добавить в него содержание для непосредственной работы. Это можно будет сделать после обновления кода по модели.

Так как этот процесс происходит не мгновенно, то можно обновить только код класса CGrennhouseView.

После обновления кода можно заполнить полученный шаблон следующим образом.

В конструкторе обнуляем переменную таймера.

```
CGrennhouseView:: CGrennhouseView()
{
m_TimerID=0;
}
```

Message заполняем едуюЩИМ образом.

```
void CGrennhouseView:Message(LPCTSTR lpszMessage)
{
CString strTemp=lpszMessage;
strTemp+=_T("\r\n");
int len=GetWindowTextLength();
GetEditCtrl().SetSel(len,len);
GetEditCtrl().ReplaceSel(strTemp);
}
```

Данная операция будет просто выводить в главное окно программы переделанную ей строку, прибавляя символы перевода строки.

Включим инициализацию таймера в операции OnWorkStartplan. Заметьте, что остановить выполнение плана можно, повторно выбран тот же пункт меню. В данном случае таймер установлен на обновление раз в секунду (второй параметр библиотечной функции SetTimer позволяет задавать интервал в миллисекундах).

```
void CGrennhouseView:: OnWorkStartplan ()
{
if (m_TimerID!=0){
        KillTimer(m_TimerID);
        m_TimeID=0;
        Message("Stop Plan");
} else {
m_TimerID=SetTimer(1,1000,NULL);
Message("Start Plan");
}
}
```

Заодно проверим как работает операция Message. Изменим операцию DestroyWindow:

```
BOOL CGrennhouseView::DestroyWindow ()
{
if (m_TimerID!=0){
        KillTimer(m_TimerID);
        m_TimeID=0;
} return CEditView:: DestroyWindow (); }
```

И, собственно, для обработчика таймера включим сообщение о его работе

```
void CGrennhouseView::OnTimer(UINT nlDEvent)
{
        Message (“Timer”);
CEditView:: OnTimer(nlDEvent);
}
```

Внесите указанные изменения и создайте исполняемый файл. Если возникли ошибки, исправьте их и запустите программу. Выберите пункт меню Work, при этом в окно программы начнут выводиться сообщения о том, что таймер работает. Остановите работу таймера, выбрав еще раз пункт Work, после этого экран программы должен выглядеть как на рис.118.

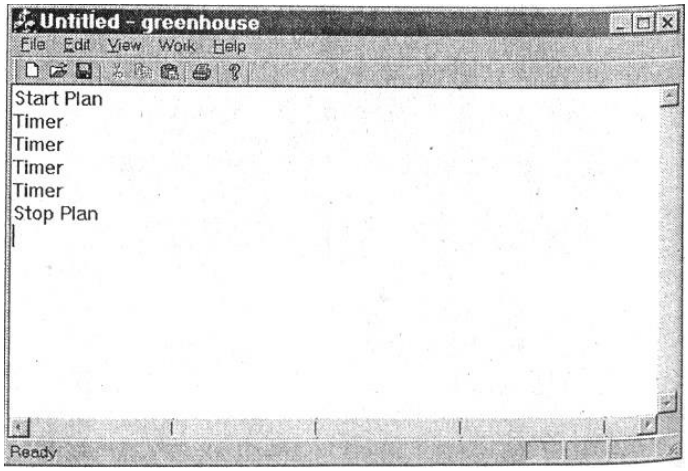

Рис.118.Экран приложение после запуска и останова работы плана