

# Глава 18. Добавление функциональности в класс документа

## Создание структуры Condition

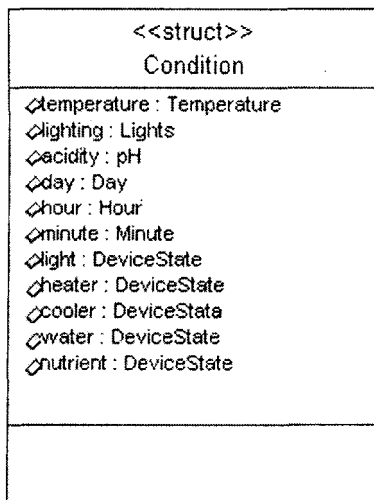
Следующим кандидатом для изменения будет класс CGreenhouseDoc. В нем будут храниться основные данные по устройствам и состоянию датчиков.

Для того чтобы этот класс мог воздействовать на среду, дополним структуру Condition дополнительными параметрами. Теперь эта структура должна будет моделировать «реальное» состояние среды, из которого датчики будут считывать показания. Также в этой структуре будет храниться «реальное» состояние исполнительных устройств (включено/выключено) и текущее время выращивания — день, час, минута.

Перейдите в диаграмму классов и создайте структуру Condition так, как показано на рис. 18.1.

**Рис. 18.1**

Структура Condition



<k

## Создание модели среды

Следует заметить, что во время работы теплицы датчики должны считывать реальную информацию, которая изменяется в зависимости от изменения окружающей среды. В нашей программе нет датчиков, с которых можно считывать реальную информацию, поэтому необходимо создать класс, который будет вносить некоторое возмущение в параметры среды.

Этот класс будет моделировать повышение или понижение температуры, а также изменение других параметров.

Создадим класс CEnvironment с операциями Change и Update. Первая будет служить для внесения Случайных возмущений в среду, а вторая будет служить для занесения текущего состояния устройств и показаний датчиков в структуру Condition после их изменения при помощи контроллера среды. Для того чтобы отражать изменения среды на экране, создадим класс CLog, который будет заниматься именно этим делом, и включим его в класс документа.

## Создание связей класса документа

Также создадим класс таймера, который будет отслеживать текущее состояние времени выращивания.

Соединим при помощи связи Unidirectional Association классы, которые мы хотим поместить в класс документа, и получим рис. 18.2.

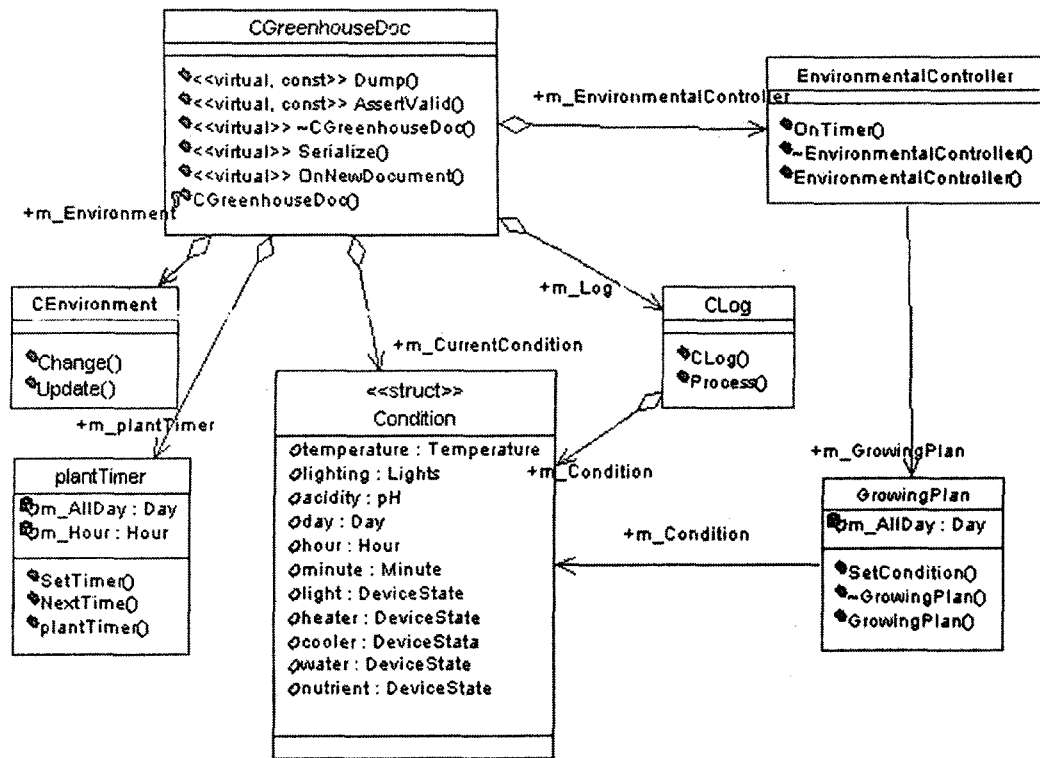


Рис. 18.2. Связи класса CGreenhouseDoc

Заметьте, что все классы, кроме GrowingPlan, включаются как агрегаты, а для структуры Condition, включенной в план выращивания, мы предусмотрели неагрегированную связь специально, для того чтобы можно было динамически создавать массив состояний на несколько указываемых при создании массива дней.

В классе CGreenhouseDoc все операции уже созданы при генерации исходного текста Мастером создания приложения VC++, и нет необходимости здесь их менять.

## Добавление функциональности

Создадим исходный текст полученных классов и добавим в него необходимую функциональность. Для этого обновим код по модели и перейдем в проект greenhouse среды Visual Studio.

Структура Condition должна выглядеть примерно так.

```
///  
struct Condition  
{  
public:  
    ///  
    Day day;  
    ///  
    Hour hour;  
    ///  
    Minute minute;  
    ///  
    Temperature temperature;  
    ///  
    Lights lighting;  
    ///  
    pH acidity;  
    ///  
    DeviceState light;  
    ///  
    DeviceState heater;  
    ///  
    DeviceState cooler;  
    ///  
    DeviceState water;  
    ///  
    DeviceState nutrient;  
};
```

Добавим в программу инициализацию плана выращивания. Для этого преобразуем класс CGrowingPlan, как показано на следующем листинге. Сам класс выглядит после генерации таким образом.

```
class CGrowingPlan  
{  
private:  
    ///  
    Day m_AllDay;  
public:  
    ///  
    Condition * m_Condition;  
    GrowingPlan(Day day);  
    GrowingPlan();  
    ///  
    SetCondition(Day, Condition * condition);  
};
```

Переменная m\_AllDay показывает нам, на сколько дней рассчитан план выращивания. Добавим содержание в тело класса и получим следующее.

```
///  
GrowingPlan::GrowingPlan(Day  
day) {  
    m_AllDay=day;  
    m_Condition=new Condition[day];  
}
```

```

//ffftModelId=3A81B6D1000A GrowingPlan() { delete[] (n_Condition);
}

//ffftModelId=3A81B6D001CF
GrowingPlan::SetCondition(Day day,Condition * condition)
{
    for (unsigned int i=day; i< m_AllDay; i++){
        ni_Condition[i].acidity=condition->acidity;
        m_Condition[i].temperature=condition->temperature;
    }
}

```

При создании класса в конструктор передается количество дней плана, для которого динамически создается необходимый массив структуры Condition.

Для заполнения структуры используем операцию SetCondition. Эта операция предназначена для заполнения значениями эталонной структуры плана выращивания. В структуре нас интересуют только эталонные состояния температуры и pH. Конечно, для того чтобы уменьшить расход памяти, можно было бы создать уменьшенный вариант только с двумя полями, но для простоты работы будем использовать ту же структуру Condition, которая используется во всей программе.

Необходимо помнить, что первым индексом в массиве будет 0. Это необходимо учитывать при проверке в классе на истечение времени выращивания. Класс таймера будет выглядеть следующим образом.

```

//ffftModelId=3A2011330320 class
plantTimer { protected:
    //ffftModelId=3A81B6CB0046
    Day m_AllDay;
    Hour m_Hour;
public:
    //ffftModelId=3A81B6CBOOB4
    plantTimer();
    //ffftModelId=3A201B8A03AC
    SetTimer(Day day,Hour hour);
    //ffftModelId=3A81B6CB0049
    BOOL NextTime(Condition *);
};

```

При установке таймера посредством операции SetTimer в нее передается количество дней выращивания. Мы приняли допущение, что время выращивания растений будет измеряться именно в сутках.

Также передадим час начала запуска плана, для того чтобы можно было бы закончить работу в то же время, но через определенное количество суток.

И тело класса преобразуем следующим образом:

```

plantTimer: : plantTimerC) {
    m_AllDay=0;
    m_Hour=0;
}

//ffftModelId=3A201B8A03AC plantTimer::SetTimer(Day
day. Hour hour)
{
    ni_AllDay=day-1;
    m_Hour=hour;
}

BOOL plantTimer::NextTime(Condition * Condition) {
    Condition->nminute++;
    if (Condition->minute>59){
        Condition->hour++;
        Condition->nminute=0;
    } if (Condition->hour>24){
        Condition->day++;
        Condition->hour=0;
    }
    // истекло время выращивания
    if (Condition->day>ni_AllDay && Condition->hour>=m_Hour)
        return FALSE;
    return TRUE;
}

```

```
}
```

Заметьте, что созданный класс таймера рассчитан на работу с интервалом в одну минуту, а запускаем мы его с интервалом в 1/1000 секунды, следовательно, время в нашей теплице бежит примерно в 60000 раз быстрее, чем на самом деле (на самом деле не настолько быстрее — все зависит от минимального интервала между генерацией событий аппаратным таймером конкретного компьютера).

Теперь можно установить количество дней в плане и обращаться к тай-меру, когда необходимо указать, что наступило время обновления. Заметим, что для обновления таймера используется внешний вызов операции NextTime, так как данный класс не имеет собственного обработчика очереди сообщений. Конечно, можно создать полностью самостоятельный таймер, который будет взаимодействовать с системой, но в данном случае это только усложнит пример, не давая дополнительных возможностей для обучения.

Класс CLog позволяет вести протокол состояния среды и исполнительных устройств. В операцию Process передается указатель на структуру Condition и указатель на класс CGreenhouseView, для того чтобы вызвать операцию Message данного класса.

```
class CLog
{
private:
public:
    //fftiM.)( I (i=3A81B6C5 0244
    void Process(Condition * condition,CGreenhouseView * pView);
//ffffModel M^3AB-Ш6C6O-Ш6
    Condition m_Condition;
//ff#ModelId=3A81B6C50212 CLogO;
};
```

Тело класса после преобразования выглядит следующим образом. В конструкторе устанавливается начальное состояние устройств и состояние среды.

```
CLog::CLog() {
    (n_Condition.acidity=0;
    in_Condition.l ighting=night;
    m_Condition.temperature=0;
    ni_Condition.heater=0ff;
    in_Condition.cooler= Off;
    m_Condition.water= Off;
    m_Condition.nutrient= Off;
}
```

При запуске процесса, перед тем как вывести протокол работы на экран, мы произведем проверку на наличие изменений в параметрах среды. Это необходимо, для того чтобы не загромождать экран информацией, которая уже выведена на предыдущем шаге. Единственное изменение, которое нам в этом случае нет необходимости выводить на экран — это изменение в показаниях прошедших суток, часов и минут.

```
//ffffModelId=3A81B6C50244 void CLog::Process(Condition
* condition,
CGreenhouseView * pView) {
    if (m_Condition.acidity^condition->acidity &&
        m_Condition.lighting==condition->lighting &&
        in_Condition.temperature==condition->temperature &&
        m_Condition.heater==condition->heater &&
        fn_Condition.cooler==condition->cooler &&
        m_Condition.water==condition->water &&
        [n_Condition.nutrient==condition->nutrient&&
        m_Condition.light==condition->light)
        return;
```

```
CString Buffer,b1;
Buffer.Forniat( "Day: %02i %02i:%02i %sT:%02.2f
pH:%02.2f L:%s H:%s C:%s W:%s N:%s",
condition->day,
condition->hour,
condition->minute,
(condition->lighting==night)?^ay": "night",
condition->temperature,
condition->acidity,
(condition->lighting==On)?"ON":"OFF",
```

```

        (condition->heater==On)?"ON":"OFF".
        (condition->cooler==On)?"ON":"OFF".
        (condition->water==On)?"ON":"OFF",
        (condition->nutrient==On)?"ON":"OFF");
pView->Message(Buffer);
m_Condition.acidity=condition->acidity;
m_Condition.lighting=condition->lighting;
m_Condition.temperature=condition->temperature;
m_Condition.heater=condition->heater;
m_Condition.cooler=condition->cooler;
m_Condition.water=condition->water;
m_Condition.nutrient=condition->nutrient;
m_Condition.light=condition->light;
};

```

Теперь добавим функциональность к классу CEnvironment.

```

//ffffModelId=3A7BB6E801F4
class CEnvironment
{
public:
    //ffffModelId=3A81B6D302BC
    Update(Condition * condition,
           EnvironmentalController * controller);
    //ffffModelId=SASIBGDSOOCB
    Change(Condition * condition,
           EnvironmentalController * controller);
};

```

Каждый час этот класс будет вносить возмущения в окружающую среду, изменяя случайным образом температуру и состояние pH, а также при включенных исполнительных устройствах будет вносить изменения в состояние окружающей среды.

```

CEnvironment::Change(Condition * condition,
                     EnvironmentalController * controller)
{
    if (condition->minute==0){
        condition->temperature=condition->temperature +3-
        (int)(rand()/(RAND_MAX/6));
        condition->acidity=condition->acidity
        +2-(int)(rand()/(RAND_MAX/4));
    } if (controller->ni_Heater.get_CurrentState()==On){
        condition->temperature+=Temperature(0.1);
    } if (controller->m_Cooler.get_CurrentState()^On){
        condition->temperature-=Temperature(0.1);
    } if (controller->fn_WaterTank.get_CurrentState()==On){
        condition->acidity-=pH(0.2);
    } if (controller->m_NutrientTank.get_CurrentState()==On){
        condition->acidity+=pH(0.2);
    } condition->acidity=round(condition->acidity,1);
    condition->temperature=round(condition->temperature,1);
    Update(condition,controller);
}

//ffffModelId=3A81B6D302BC CEnvironment::Update(Condition * condition,
EnvironmentalController * controller) {
    condition->heater=controller->m_Heater.get_CurrentState();
    condition->cooler=controller->ni_Cooler.get_CurrentState();
    condition->water=controller->fn_WaterTank.get_CurrentState();
    condition->nutrient=
    controller->ni_NutrientTank.get_CurrentState();
    condition->lighting=
    (condition->hour>=9 && condition->hour<=20)?day:night;
    condition->light=controller->m_Light.get_CurrentState();
    controller->m_TemperatureSensor.set_Value(condition->temperature);
    controller->ni_pHSensor.set_Value(condition->acidity);
}

```

Теперь, когда готово все, кроме исполнительных устройств, можно посмотреть, как будет работать наше тепличное хозяйство. Для этого добавим в класс CGreenhouseDoc следующий код для инициализации начального состояния структуры Condition и задания

плана выращивания.

```
//ffftModel IC1-3A1AD7A30-IF4
```

```
BOOL CGreenhouseDoc: :OnNewDocuient()
```

```
{  
    if (! CDocument::OnNewDocument()) return FALSE;  
    m_CurrentCondition.day^O;  
    ni_CurrentCondition.hour=9;  
    fn_CurrentCondition.ion.mi nute=0;  
    in_CurrentCondition. teniperature=25;  
    [n_CurrentCondition.l ighting=day;  
    ni_CurrentCondition.acidity=7;  
    ni_CurrentCondition.light=0ff;  
    ni_CurrentCondition.ion.heater=0ff;  
    m_CurrentCondition.cooler=0ff;  
    (n_CurrentCondition.ion.water=0ff;  
    m_CurrentCondition.nutrient=0ff;  
    in_Envi roninenta l Control ler. m_Growi ngPlan->SetCond ition (0, &m_CurrentCond ition);  
    m_plantTiitier. SetTimer(1, ni_CurrentCond ition. hour);  
    return TRUE;  
}
```

И добавим обработку сообщения таймера в CGreenhouseView.

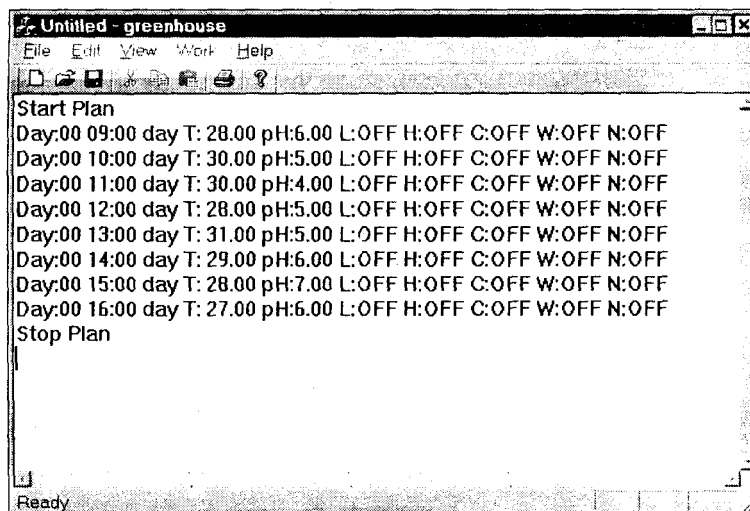
```
void CGreenhouseView: : OnTin)er(UINT nIDEvent) {
```

```
    GetDocuient( )->ni_Envi ronment. Ghange(  
        &GetDocuient( )->m_CurrentCond ition, &GetDocument( )->m_Envi ronmental  
        Controller);  
    GetDocuient( )->m_Envi ronment. Update(  
        &GetDocu[nent( )->m_CurrentCond ition, &GetDocument( )->m_Envi roninenta l  
        Control ler);  
    GetDocuient( )->m_Log. Process(  
        &GetDocu(nent( )->m_CurrentCond ition, th is);  
    if (!GetDocument( )->m_plantTimer.NextTime(  
        &GetDocument( )->in_CurrentCond ition)){ if (m_Tinier!D!=0){  
        Ki l lTinier(in_Tiiiiierl D);  
        m_TimerID = 0;  
    } GetDocument( )->ni_Log. Process(  
        &GetDocument( )->m_CurrentCondition,this);  
    MessageC'End plan");  
} CEditView: : OnTinier(n l DEvent);  
}
```

После запуска полученного приложения у вас должно получиться следующее (рис. 18.3). Все устройства выключены, поэтому температура и pH изменяются случайным образом. В нашей теплице теперь не хватает только исполнительных устройств, которые будут компенсировать изменения температуры и pH.

**Рис. 18.3**

Работа приложения без исполнительных устройств



**Обозначения** следующие:

- Day — сколько дней прошло с запуска плана выращивания, показывает текущее время. В моем случае посадки были произведены в девять утра и урожай должны будем снимать также в девять утра, но уже через указанное количество дней;
- Показывается текущее время суток day/night;
- T — текущая температура в теплице;
- pH — текущий показатель кислотности;
- L — отражает состояние осветителей;
- H — отражает состояние нагревателей;
- C — отражает состояние вентиляторов;
- W — отражает состояние крана для подачи воды. ON — вода подается;
- N — отражает состояние заслонки для подачи удобрений. ON — удобрения подаются.

## **Заключение**

В этой главе вы:

1. Создали дополнительные классы и структуры, связанные с классом документа.
2. Создали необходимые связи с классом документа.
3. Добавили необходимую функциональность в созданные классы.
4. Создали приложение, которое протоколирует изменения среды, но не оказывает на них влияния.