CropModelMKS用户手册

目录

[CropModelMKS用户手册 1](#_Toc57369228)

[绪论 2](#_Toc57369229)

[用户需求规约 3](#_Toc57369230)

[概要设计 3](#_Toc57369231)

[详细设计 4](#_Toc57369232)

[State 5](#_Toc57369233)

[Parameter 5](#_Toc57369234)

[Simulator 6](#_Toc57369235)

[Analyzer 6](#_Toc57369236)

[Model\_Module 6](#_Toc57369237)

[ILibrary 6](#_Toc57369238)

[用户手册 7](#_Toc57369239)

[图形界面操作 7](#_Toc57369240)

[组件注册 7](#_Toc57369241)

[创建参数文件 11](#_Toc57369242)

[创建项目文件 11](#_Toc57369243)

[模拟石羊河春小麦生长 11](#_Toc57369244)

[灌溉制度优化 12](#_Toc57369245)

[底层组件开发 12](#_Toc57369246)

[C++ 12](#_Toc57369247)

[MATLAB 25](#_Toc57369248)

[Python 28](#_Toc57369249)

[Fortran 30](#_Toc57369250)

[演示组件原理简介 42](#_Toc57369251)

[CropModelMKS.Climate.Default 42](#_Toc57369252)

[CropModelMKS.Plant.None 42](#_Toc57369253)

[Environment.WaterBalance.Example 42](#_Toc57369254)

[Management.Schedule.Example 42](#_Toc57369255)

[Plant.WaterDriven.Example 42](#_Toc57369256)

# 绪论

作物模型是模拟作物生长行为的计算机程序[1]，被广泛地应用在农业、环境、经济等领域的研究与工程项目中。其针对的内容包括基因型改进、作物管理、种植系统模拟及区域规划等[3]。如今，已经有超过250种作物模型被开发出来。

DSSAT,APSIM,RECORD,APES,MODCOM,CROSPAL,EPIC

Multi-Modelmethods

Recently the

开发作物模型的热潮在20世纪90年时达到了一个高潮。一般认为，开发作物模型设计到的基本原理在那是已经基本完备。时至今日，已经有超过250种作物模型被开发出来。

但在这250中作物模型中，较为常用的主要有WOFOST、SWAP、DSSAT、APSIM、STICS等一些模型。这些模型，也逐渐发展为模型系统。

此外，作物模型一般没有合适的调用方式[15]。这导致作物模型在调用时时步骤繁琐，甚至不能被调用。如今，越来越多的数学方法（例如优化，数据同化等）可以被应用在作物模型的研究中，这些数学方法将会大量循环调用作物模型进行运算。若没有合适的调用方式，作物模型仅通过读写文件的方法进行数据传输，将导致整体运算效率的降低。

因此，有必要针对作物模型在开发、改进与应用中的问题，开发一款作物模型平台，以期提升对于作物模型研究中代码的复用效率，降低作物模型使用的难度，进而提升作物模型的运算效率。

如今，使用较为广泛的模型系统有DSSAT[16]、APSIM[17]与STICS[18]。

作物模型平台的框架已经在向前发展。模块化的作物模型系统逐渐取代各平台中原有的框架[21]。模块化的模系统认为作物模型仅由模块组成，由模块负责包装模拟程序并仅能通过输入输出与外部通信[22]。ModCom是较早的模块化系统[23]。该系统基于COM技术设计各模块的接口，并在系统中集成了常微分方程的求解器。最近的被开发出来的模块化系统是CROSPAL[24]与RECORD[25]。

但是新开发出来的模块化系统较为复杂，在设计时大量依赖于其他模拟环境。比如RECORD系统基于Virtual Libraries Environment（VLE）环境编写，而VLE又是基于DEVS环境搭建而成。DEVS指离散系统事件规范[26]，系统中包括基本模型（atomic model）与耦合模型（coupled model）。了解这个系统的细节以及操作流程就会花费很多精力，更遑论后面对于VLE与RECORD本身的学习。

随着第四范式的兴起[28]，数据驱动的理念又对作物模型平台的开发带来了新的挑战。对于新一代作物模型平台的讨论认为[29]，新一代的作物模型平台的开发过程应该更多地遵循软件工程的规范，并且在开发时应考虑高性能计算、人工智能等新兴信息技术的影响。同时应该注意对不同数据源的兼容。

CropModelMKS旨在减小作物模型在开发、改进与使用过程中的负担，增强作物模型相关工作的共享效率。

必须承认传统程序猿与科研人员在编程语言与计算机知识上的差异。因此，本文在开发平台的同时也对利用不同编程语言进行了测试。为不同编程背景的人员提供了开发的模板。

CropModelMKS可以从[Welcome to Crop Model MKS | Crop Model MKS A Micro kernel for the Crop Models (songjim624.github.io)](https://songjim624.github.io/CropModelMKS.github.io/)获取。

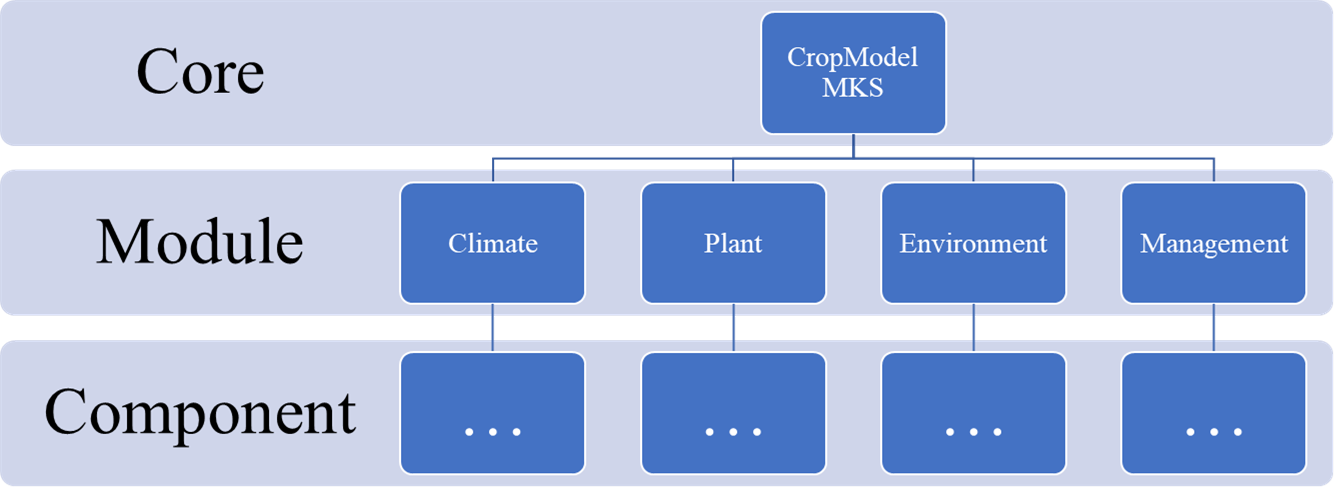
# 用户需求规约

本产品的用户需求主要产生于软件作者与同学、同事在作物模型的使用与改进过程中所与遇到的不便，这些问题历来被作物模型的开发者、使用者与维护者所诟病。因此，CropModelMKS旨在解决由作物模型的结构所导致的其在开发、使用及维护过程中所遇到的问题。

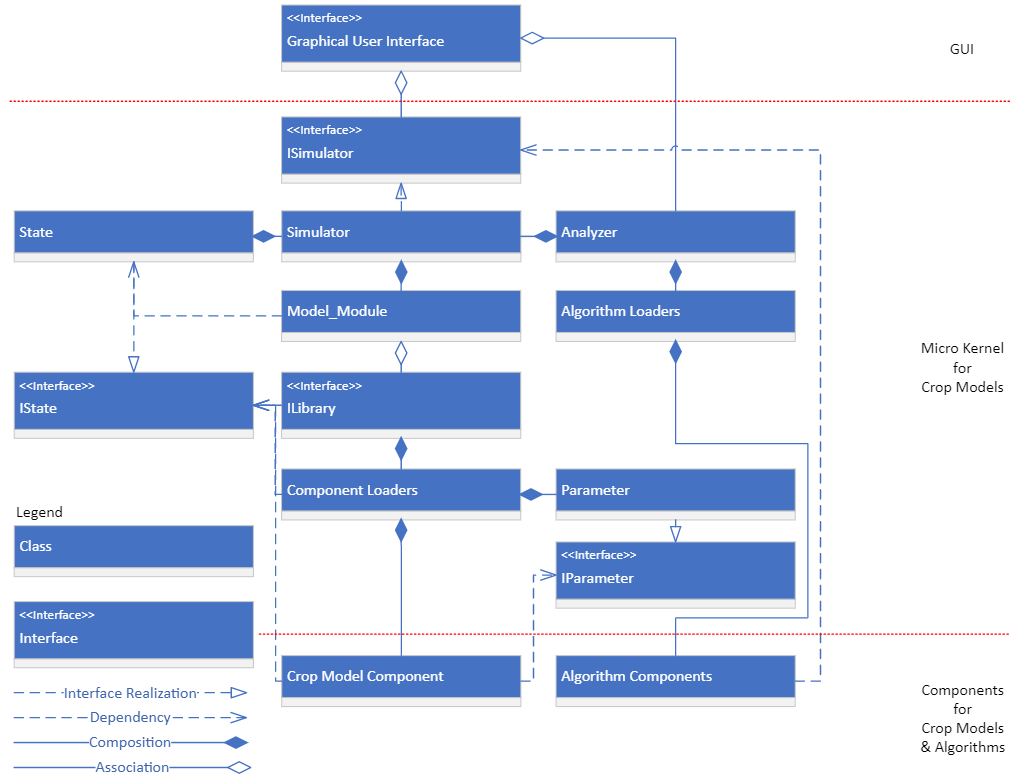
运行平台Windows10、.Net Framework 4.7.2，依赖库PyQt5(version 15.0)，Pythonnet

# 概要设计

Crop ModelMKS的主要结构由内核、模块与组件三个层次组成，如图所示：



其中，模块只有气象、植物、环境与管理四种不同的类型。



本产品核心为作物模型的微内核系统，该系统主要由State，Parameter，Simulator与Analyzer组成。在该体系下，作物模型的模拟功能全部封装在底层组件之中，组件在运行时的生命周期由Simulator维护，组件的注册则由GUI负责。

为了实现非侵入式的架构。

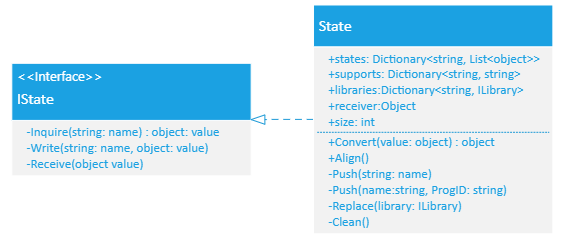
本平台的非侵入式架构主要依托COM技术实现。模块间的通信主要通过名为State与Parameter的COM组件。

在设计Simulator时采用了.Net Framework 4.7.2.，因为在.Net Framework中，类可以通过ComVisible模拟为COM组件，便于Simulator对于State和Parameter的管理与维护。

由于平台支持多种编程语言编译而成的库，底层库（即Crop Model Components与Algorithm Components）既可以从已有的作物模型迁移而来，也可以自行开发。

# 详细设计

## State

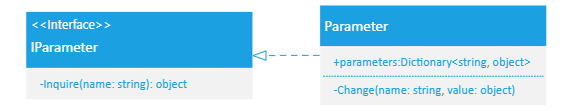


State类（后文以**State**表示）是储存模拟结果的组件，也是实现作物模型各组件间数据交换的重要通道。作物模型所有的模拟结果都被储存在该类的states字典中。为了实现各模块间的数据交互，**State**同时维护一个supports字典与libraries字典。Supports字典中储存所要查询的变量在哪个组件中，libraries字典则储存组件名称与组件相应的ILibrary接口。

为了避免对**State**进行不当的操作，**State**的功能被划分为两个部分。Simulator可以调用**State**暴露出来的Push、Replace与Clean等功能对State内储存的字典进行操作。而各模块只能通过**State**抽象出的接口IState进行数据交换。也就是说，Simulator可以对State内部字典的结构进行更改，也可以查询或者清除**State**内部的数据，但底层的模块不可以，各模块只能向**State**查询或写入数据。

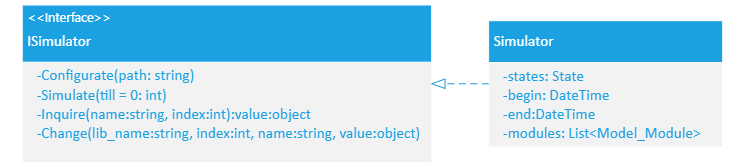
模块可以通过Inquire函数向**State**查询数据，可以通过Write函数向**State**写入模拟结果，通过Receive函数将此模块支持的变量发送个其他模块。当模块通过Inquire函数向**State**请求数据时，Inquire函数的内部流程如图所示：

## Parameter



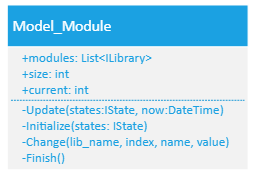
Parameter类储存模块模拟时所需要的参数。一个模块所需的所有参数都被储存在parameters字典中，parameters的键为参数的名称，值为对应参数的值。Parameter类向Simulator暴露了一个Change方法，Simulator可以通过该方法改变Parameter中某一参数的值。而底层的各模块则只能通过IParamter接口提供的Inquire方法向Parameter查询数据。

## Simulator



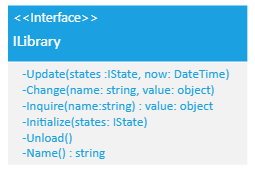
## Analyzer

## Model\_Module



一个模块下面，按时间段的不同，可能会调用不同的组件。因此，组件信息按照调用顺序的先后储存在modules中。size表示modules的长度，current表示当前正在被使用的组件在modules中的位置。

## ILibrary



ILibrary统一了不同语言制作而成的底层组件在Simulator中的调用方法。图1-中的Loaders则是对ILibrary的不同实现。

Library\_Default

是默认的组件加载功能，支持的组件类型包括C++与Python。

对于Fortran而言，由于Intel Visual Fortran中并不能得到函数的返回值，Fortran组件将通过包装后的State与Parameter传输数据。

Libraray\_MATALB

MATLAB则通过

# 用户手册

基于COM技术的作物模型组件采用统一的接口，其形式为：

interface IComponent : IDispatch

{

[id(1)] HRESULT Update([in] IDispatch\* states);

[id(2)] HRESULT Initialize([in] IDispatch\* states, [in] IDispatch\* parameters);

[id(3)] HRESULT Inquire([in] BSTR name, [in] IDispatch\* states);

[id(4)] HRESULT Description([in] BSTR path);

[id(5)] HRESULT Convert([in] BSTR file\_path, [in] BSTR xml\_path);

};

基于该技术的分析方法组件也采用统一的接口，其形式为：

interface IComponent : IDispatch

{

[id(1)] HRESULT Analyze([in] IDispatch\* simulator);

[id(2)] HRESULT Configurate([in] BSTR analyzer, [in] BSTR simulator);

[id(3)] HRESULT Inquire([in] BSTR name, [in] IDispatch\* states);

[id(4)] HRESULT Description([in] BSTR path);

[id(5)] HRESULT Convert([in] BSTR file\_path, [in] BSTR xml\_path);

};

基于.Net Assembly的作物模型组件采用统一的类，其形式为：

主要针对使用人员与维护人员，主要内容为界面的使用以及底层库的开发模板。

# 图形界面操作

本章将以用例的形式来介绍CropModelMKS的操作流程。

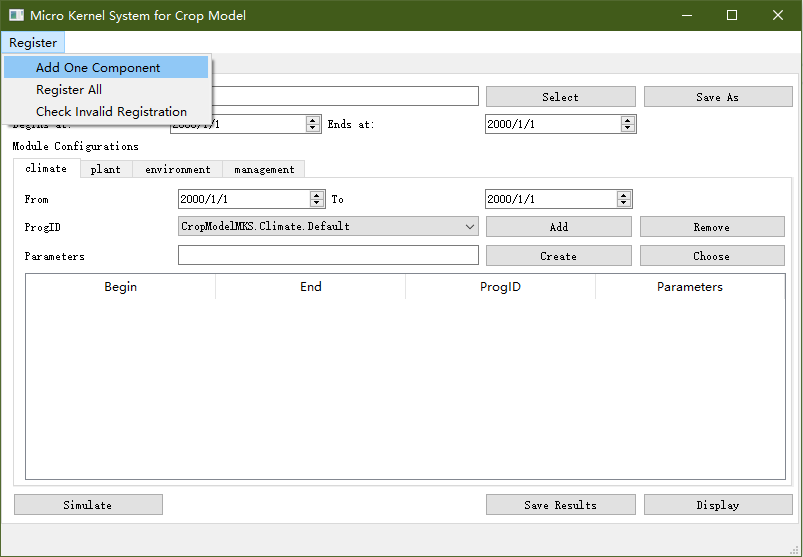
## 组件注册

组件注册：以注册组件CropModelMKS.Plant.None为例

开发好的组件需要先在CropModelMKS中注册，然后CropModelMKS才能识别该组件。

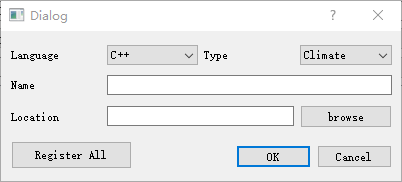
注册组件的流程为：

如下图所示：



点击菜单栏中的Register按钮，将会弹出下拉菜单。

点击下拉菜单中的Add One Component按钮，将会弹出如下所示对话框。

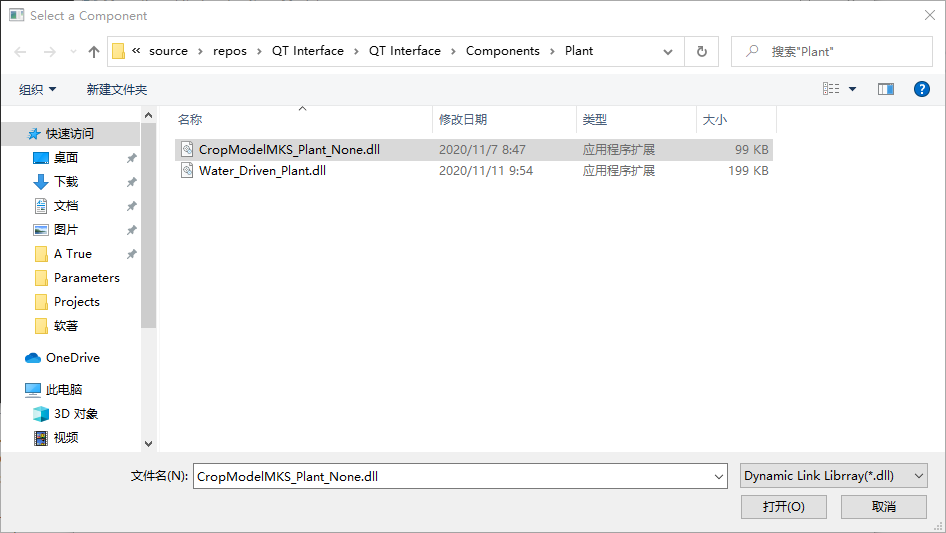


该对话框中的Language表示制作该组件时所使用的编程语言，CropModelMKS支持的编程语言有C++、Fortran、MATLAB、C#、Python以及R。其中Type表示该组件负责实现哪一个模块的功能，其种类包括Climate、Plant、Environment、Management与Analysis。Name表示该组件的ProgID。Location表示该组件库文件的位置。

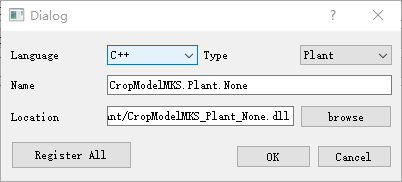
为了方便管理组件，所有的组件都应储存在CropModelMKS安装目录下的Components文件夹中。Components文件夹中又包含Climate、Plant、Environment、Management与Analysis等子文件夹，负责储存具体各模块的组件。

Name需要手动输入，而库文件的位置则可以通过browser按钮进行选择。

在点击browser按钮之后，组件注册模块将会根据Type类型自动打开相应的文件夹，如图所示。

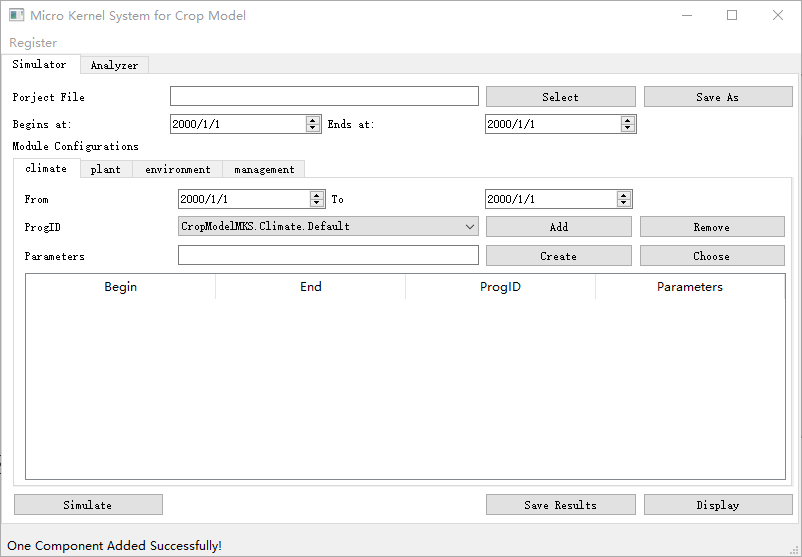


在其中选择CropModelMKS\_Plant\_None.dll，然后再点击打开。将回到组件注册界面，如图所示。



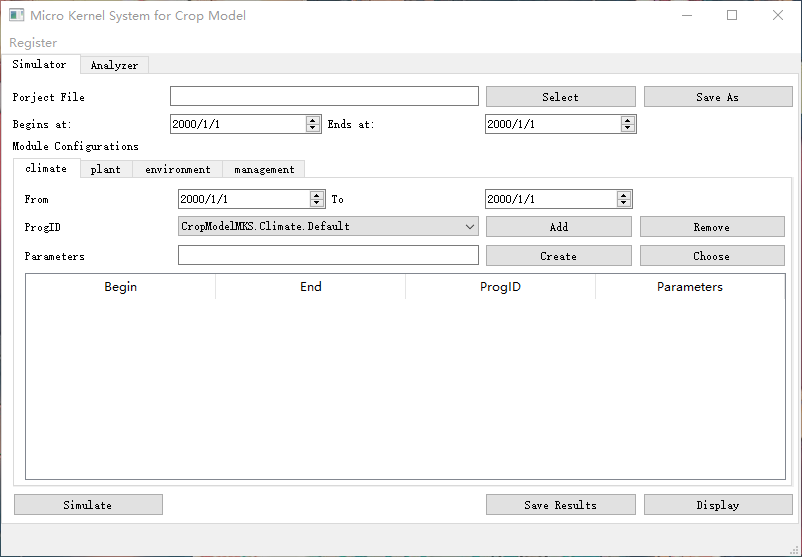
此时点击OK按钮，完成注册。

组件注册成功后，将会在主界面的状态栏进行显示。



组件注册的内部过程：

数据准备



## 创建参数文件

## 创建项目文件

## 模拟石羊河春小麦生长

项目文件的结构如下所示：

<?xml version="1.0" encoding="UTF-8"?>

<Configuration>

<Clock>

<Begin>2014/3/26</Begin>

<End>2014/7/1</End>

</Clock>

<Module type="climate">

<Library>

<language>C++</language>

<ProgID>CropModelMKS.Climate.Default</ProgID>

<location></location>

<begin>2014/3/26</begin>

<end>2014/7/1</end>

<parameters type="Custom">C:/Users/Song Jim/source/repos/QT Interface/QT Interface/Projects/Climate-Light.CLI</parameters>

<converted>C:/Users/Song Jim/source/repos/QT Interface/QT Interface/Projects/A True/Parameters/Climate-Light.CLI.XML</converted>

</Library>

</Module>

<Module type="plant">

<Library>

<language>Plant</language>

<ProgID>Plant.WaterDriven.Example</ProgID>

<location></location>

<begin>2014/3/26</begin>

<end>2014/7/1</end>

<parameters type="Custom">C:/Users/Song Jim/source/repos/QT Interface/QT Interface/Projects/Plant.CRO</parameters>

<converted>C:/Users/Song Jim/source/repos/QT Interface/QT Interface/Projects/A True/Parameters/Plant.CRO.XML</converted>

</Library>

</Module>

<Module type="environment">

<Library>

<language>C++</language>

<ProgID>Environment.Balance.Example</ProgID>

<location></location>

<begin>2014/3/26</begin>

<end>2014/7/1</end>

<parameters type="Custom">C:/Users/Song Jim/source/repos/QT Interface/QT Interface/Projects/Test.SOL</parameters>

<converted>C:/Users/Song Jim/source/repos/QT Interface/QT Interface/Projects/A True/Parameters/Test.SOL.XML</converted>

</Library>

</Module>

<Module type="management">

<Library>

<language>C++</language>

<ProgID>CropModelMKS.Management.Default</ProgID>

<location></location>

<begin>2014/3/26</begin>

<end>2014/7/1</end>

<parameters type="Custom">C:/Users/Song Jim/source/repos/QT Interface/QT Interface/Projects/NoIrr.MAN</parameters>

<converted>C:/Users/Song Jim/source/repos/QT Interface/QT Interface/Projects/A True/Parameters/NoIrr.MAN.XML</converted>

</Library>

</Module>

</Configuration>

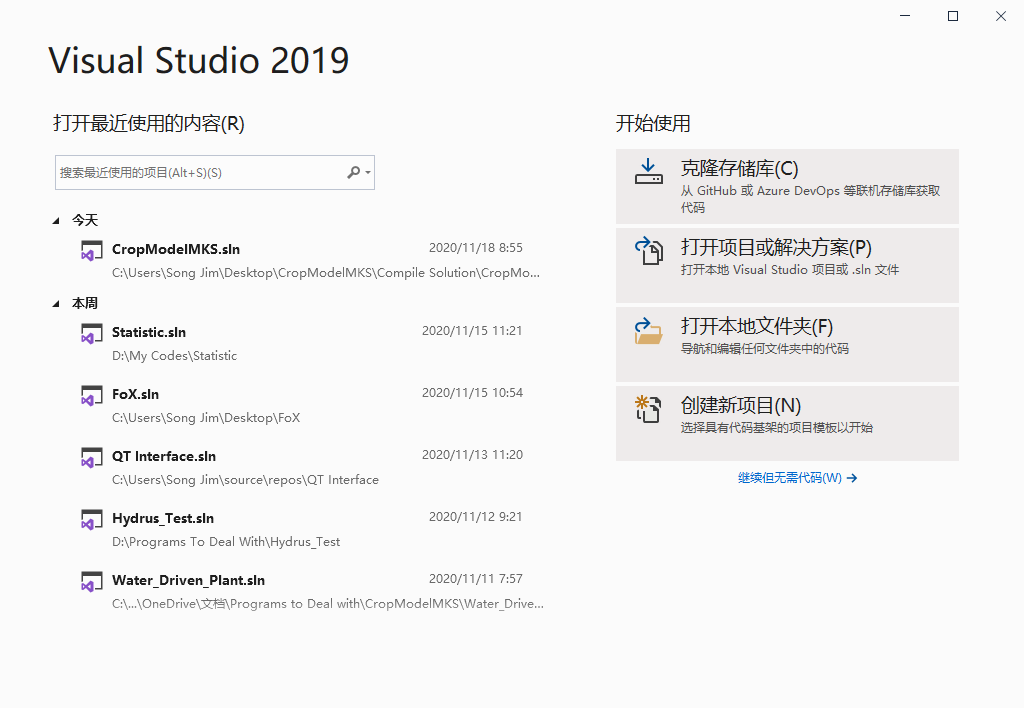
## 灌溉制度优化

# 底层组件开发

### C++

利用C++构建作物模型的模块组件，以作物模块为例

打开Visual Studio2019，选择创建新项目



点击后，选择其中的ATL项目

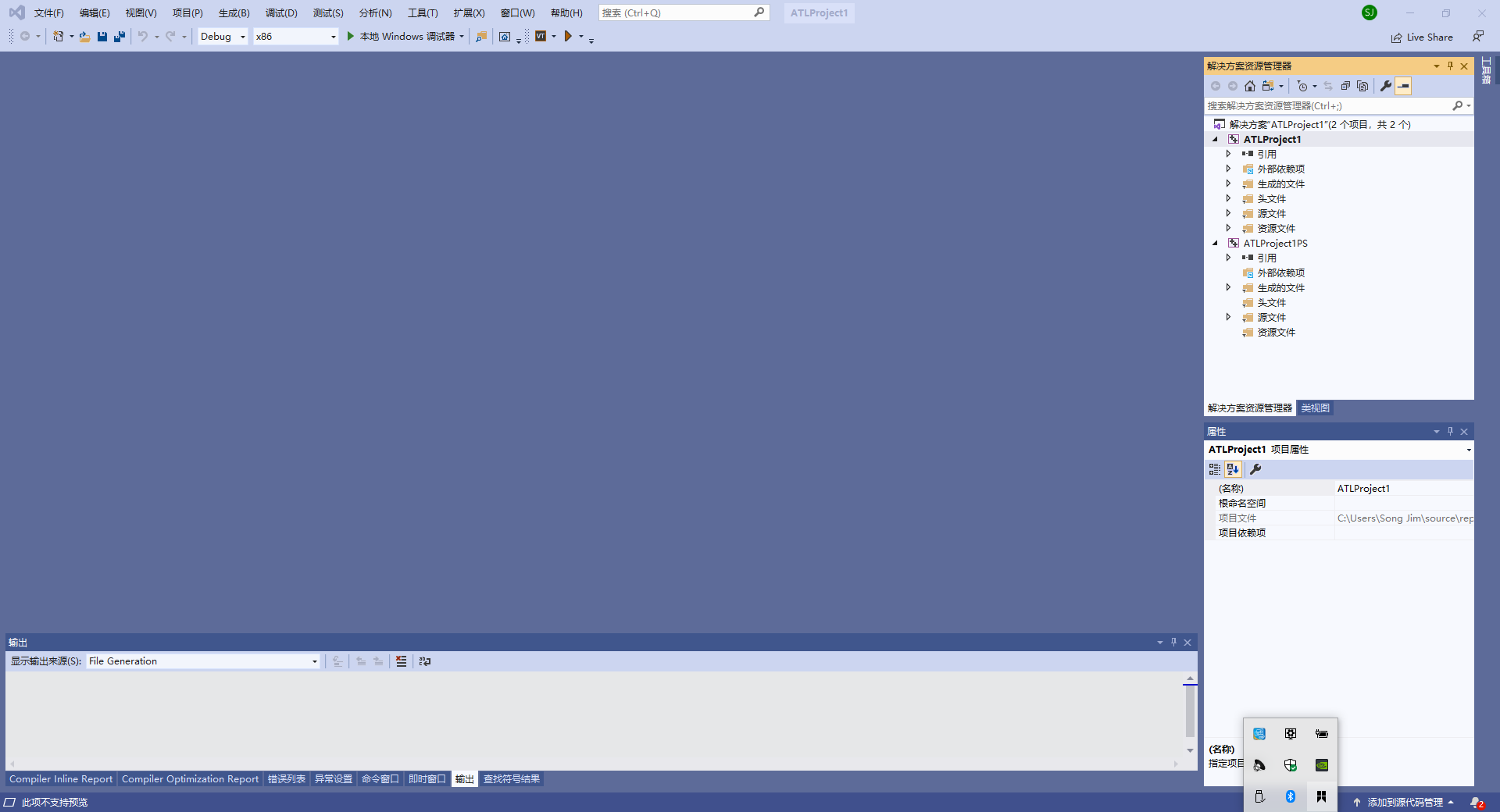


创建时，类型选择动态链接库，注意勾选“支持COM+1.0（T）”与“支持组件注册”



点击确定，生成解决方案。

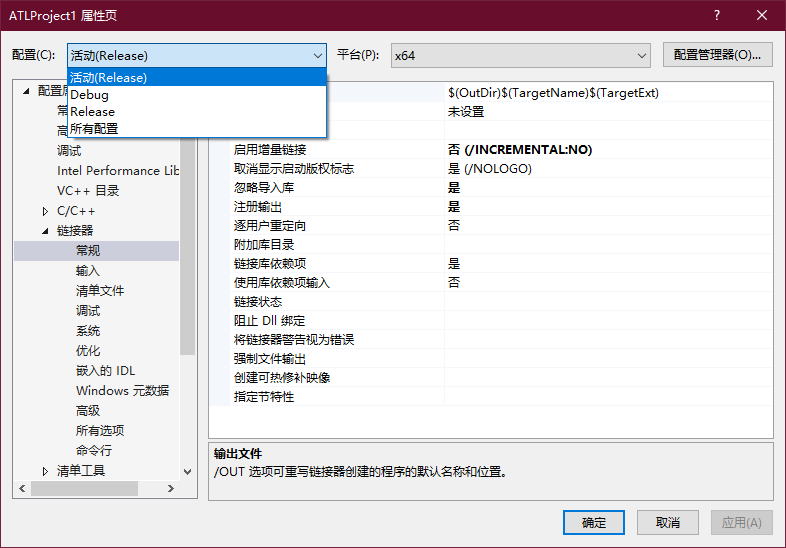
生成解决方案后，首先更改解决方案的配置。



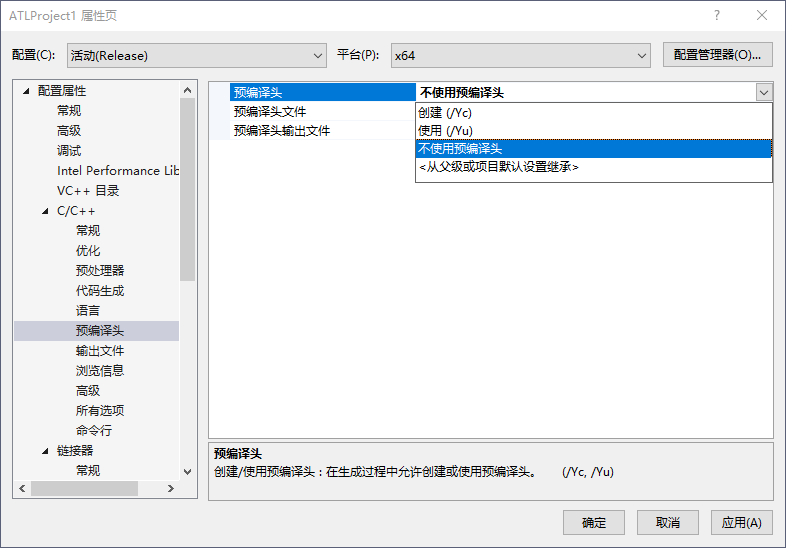
右键点击解决方案，选择其中的属性。



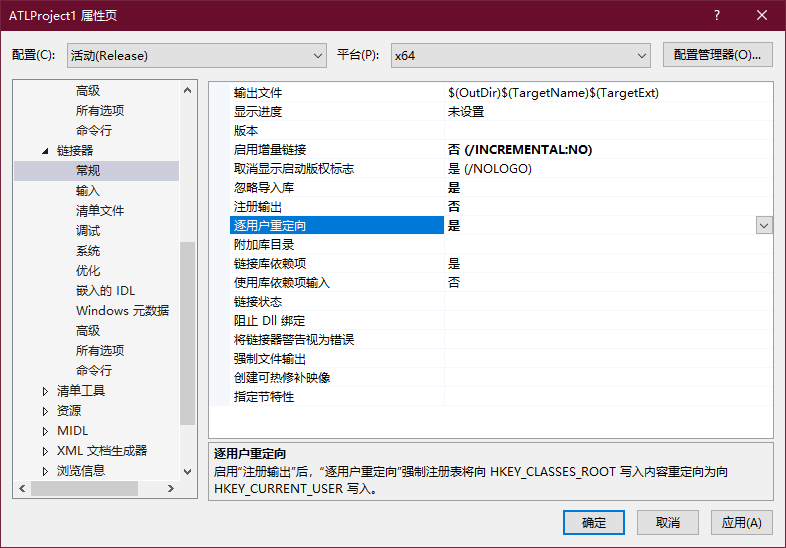
配置改为Release，平台改为x64。



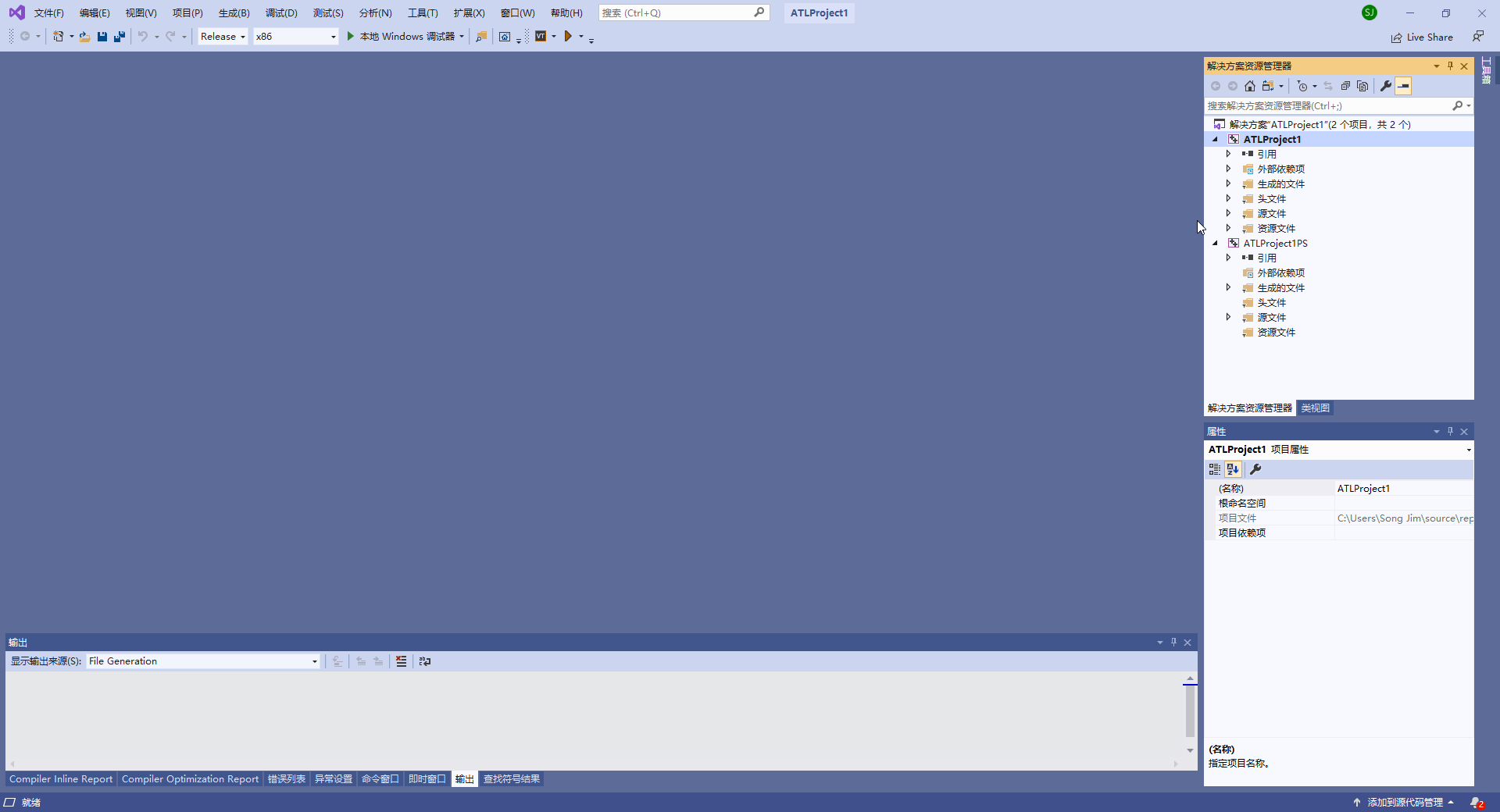
将C/C++的预编译头配置改为“不使用预编译头”。



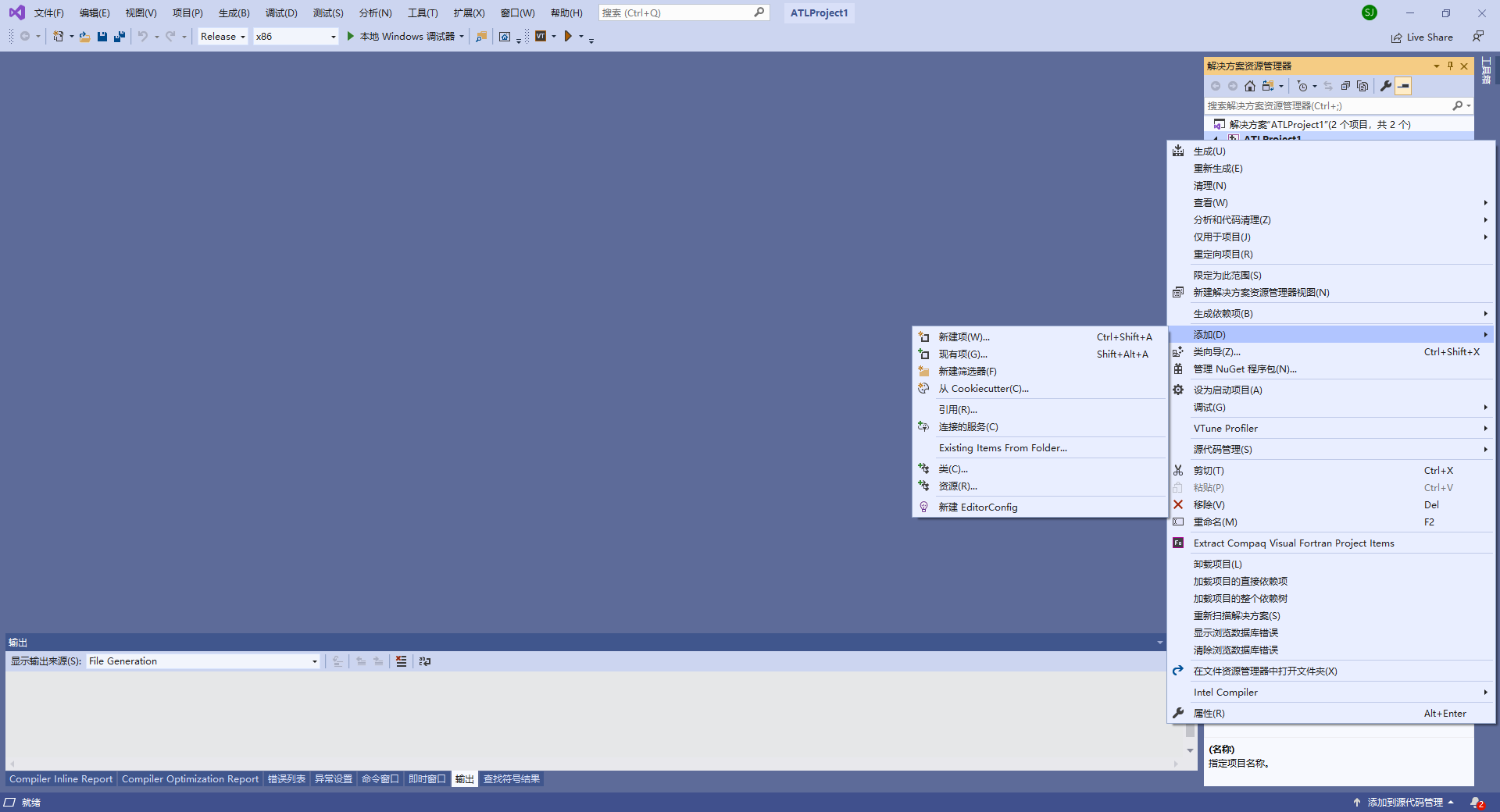
将链接器-常规中的注册输出改为“否”，逐用户重定向改为“是”



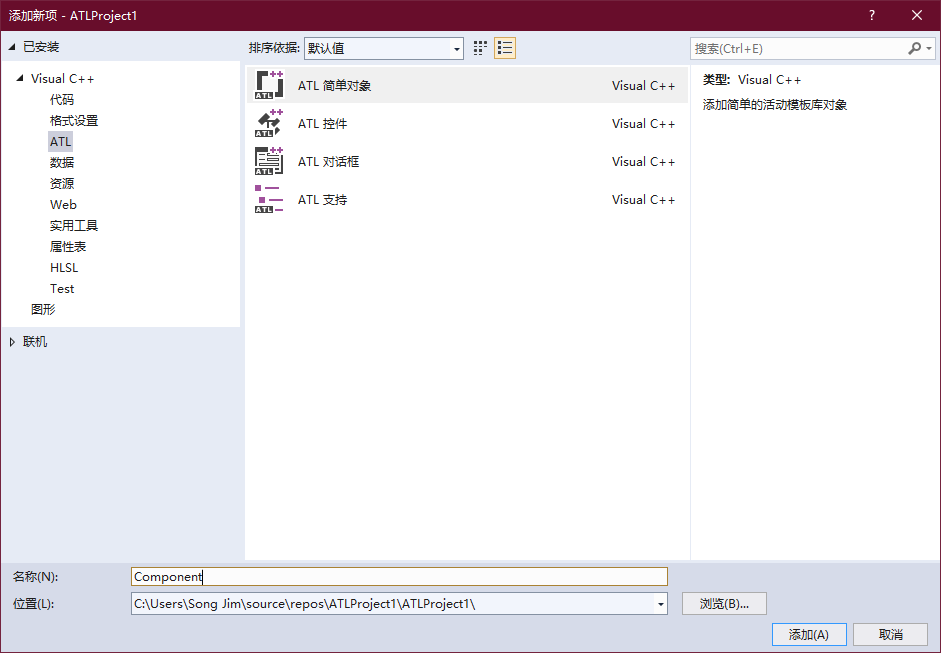
点击应用，然后对项目ATLProject1PS执行同样的操作。



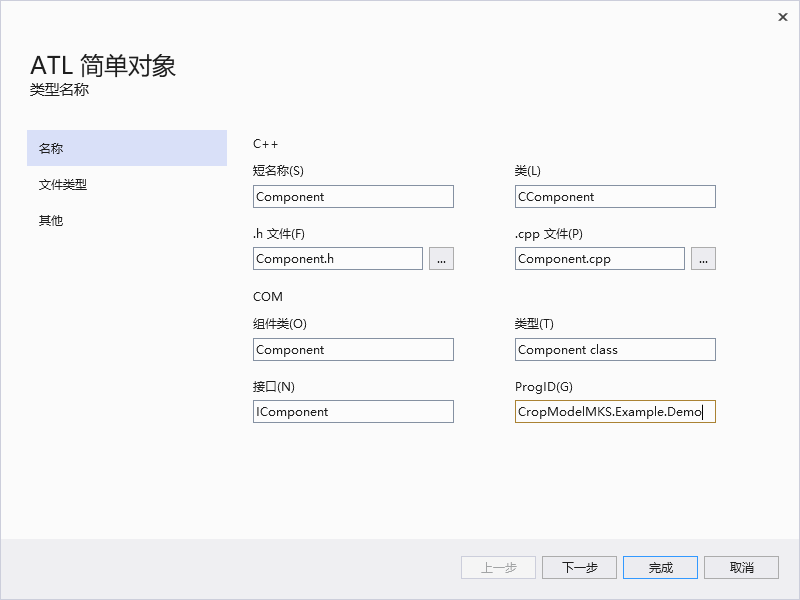
然后右键点击项目ATLProject1，选择添加-新建项。



选择简单ATL对象，推荐将ATL对象命名为Component，点击添加。



设置好ATL的ProgID之后，即可直点击“完成”。



完成后，在Component.h中添加对于头文件string.h、comutil.h以及组件核心Core.h的引用，即：

#include<string>

#include<comutil.h>

#include "×××\Core.h"

并且链接库comsuppw.lib与kernel32.lib

#pragma comment(lib, "comsuppw.lib")

#pragma comment(lib, "kernel32.lib")

如图所示



然后向类CComponent中添加内容：

private:

Core<float>\* core;

public:

virtual HRESULT \_stdcall Initialize(IDispatch\* istates, IDispatch\* iparameters)

{

core = new Core<float>(CComDispatchDriver(istates),CComDispatchDriver(iparameters));

return S\_OK;

}

virtual HRESULT \_stdcall Update(IDispatch\* istates)

{

return core->Update(ATL::CComDispatchDriver(istates));

}

virtual HRESULT \_stdcall Inquire(BSTR name, IDispatch\* istates)

{

std::string n = \_bstr\_t(name);

return core->Inquire(n, ATL::CComDispatchDriver(istates));

}

virtual HRESULT \_stdcall Description(BSTR path)

{

Describe<float>(std::string(\_bstr\_t(path)));

return S\_OK;

}

virtual HRESULT \_stdcall Convert(BSTR file\_path, BSTR xml\_path)

{

return ::Convert(std::string(\_bstr\_t(file\_path)), std::string(\_bstr\_t(xml\_path)));

}

如图所示：



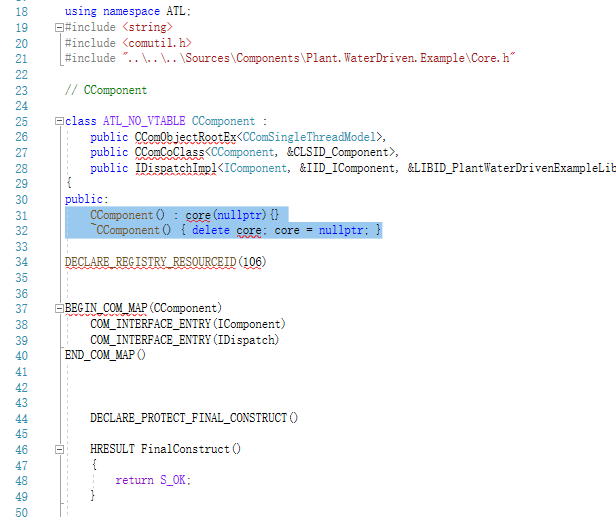
并将类的构造函数改为：

CComponent() : core(nullptr){}

增加析构函数：

~CComponent() { delete core; core = nullptr; }

如图所示：



最后，向源文件中的.idl文件中添加如下内容：

[id(1)] HRESULT Update([in] IDispatch\* states);

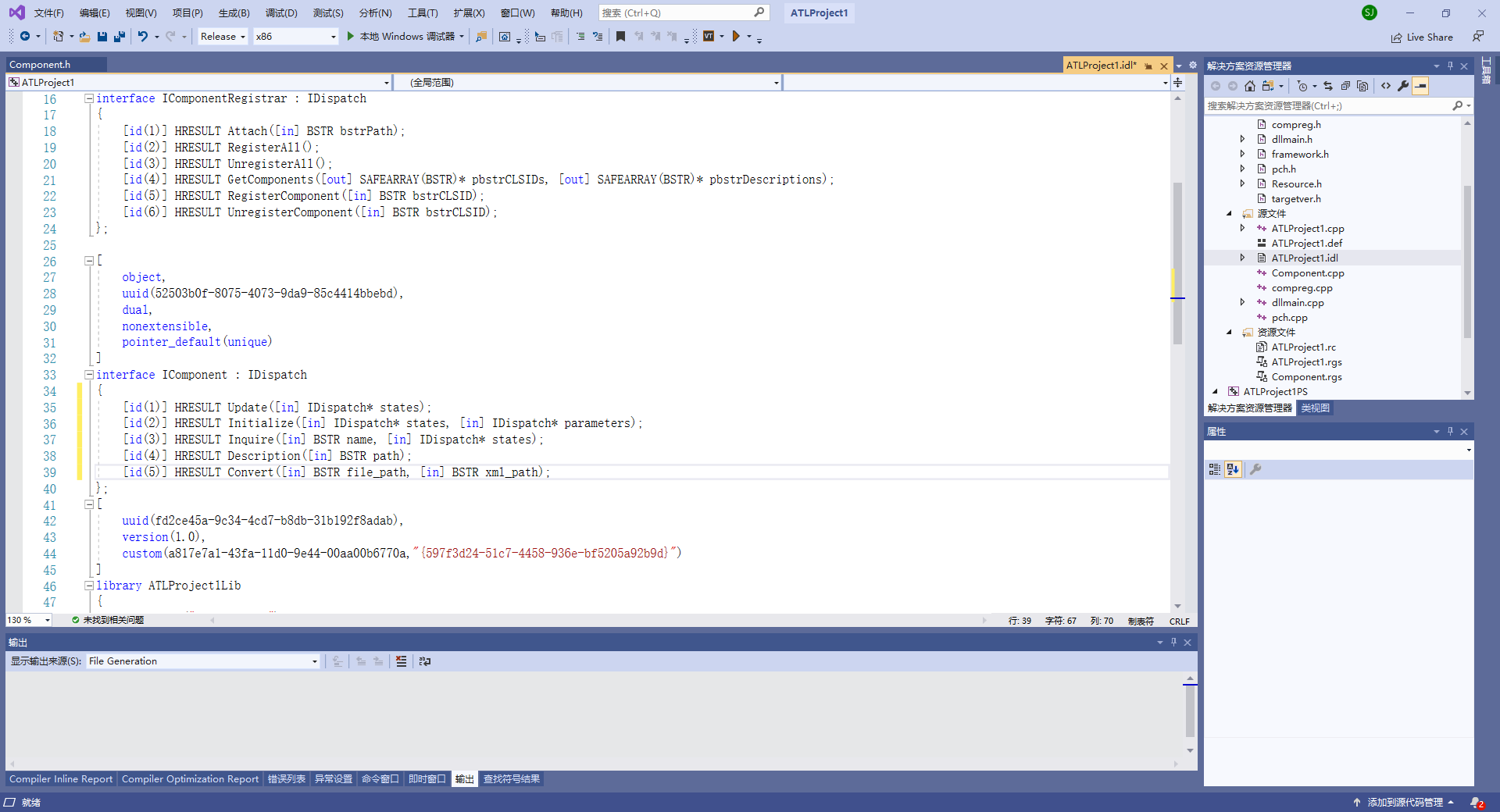
[id(2)] HRESULT Initialize([in] IDispatch\* states, [in] IDispatch\* parameters);

[id(3)] HRESULT Inquire([in] BSTR name, [in] IDispatch\* states);

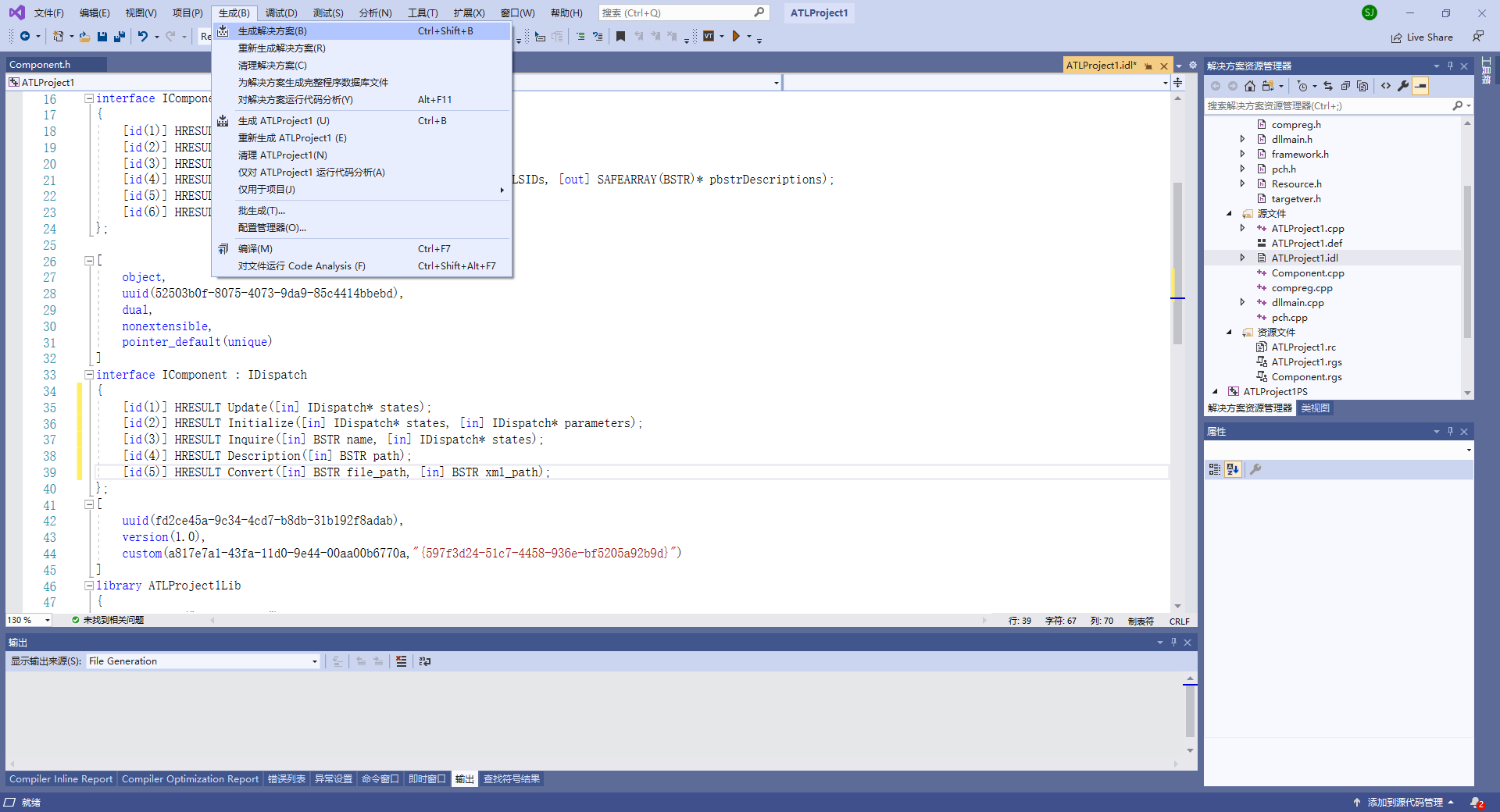
[id(4)] HRESULT Description([in] BSTR path);

[id(5)] HRESULT Convert([in] BSTR file\_path, [in] BSTR xml\_path);

如图所示：

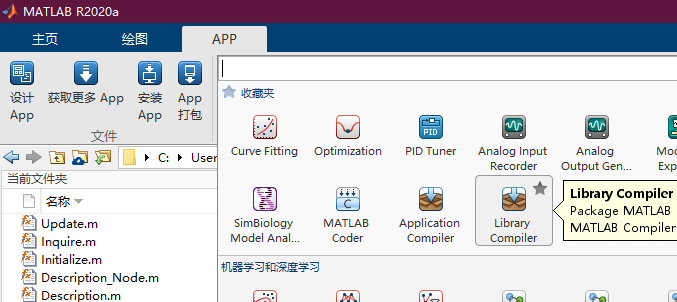


最后编译生成动态链接库即可。

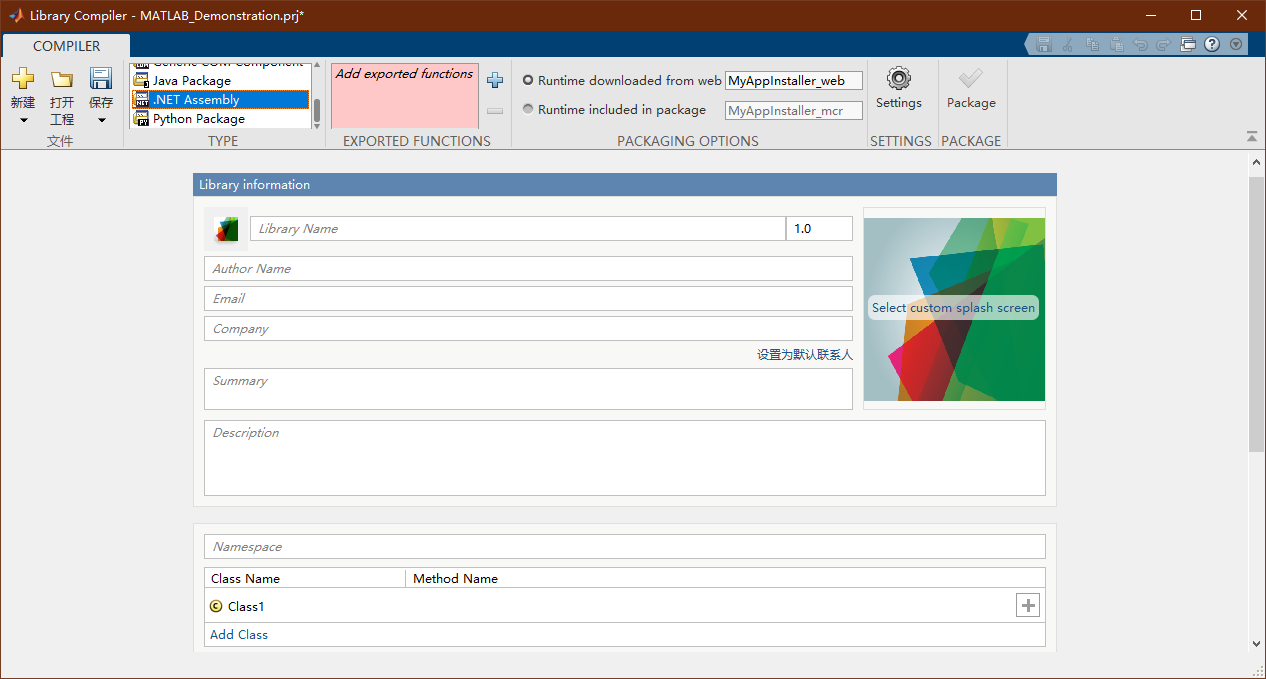


### MATLAB

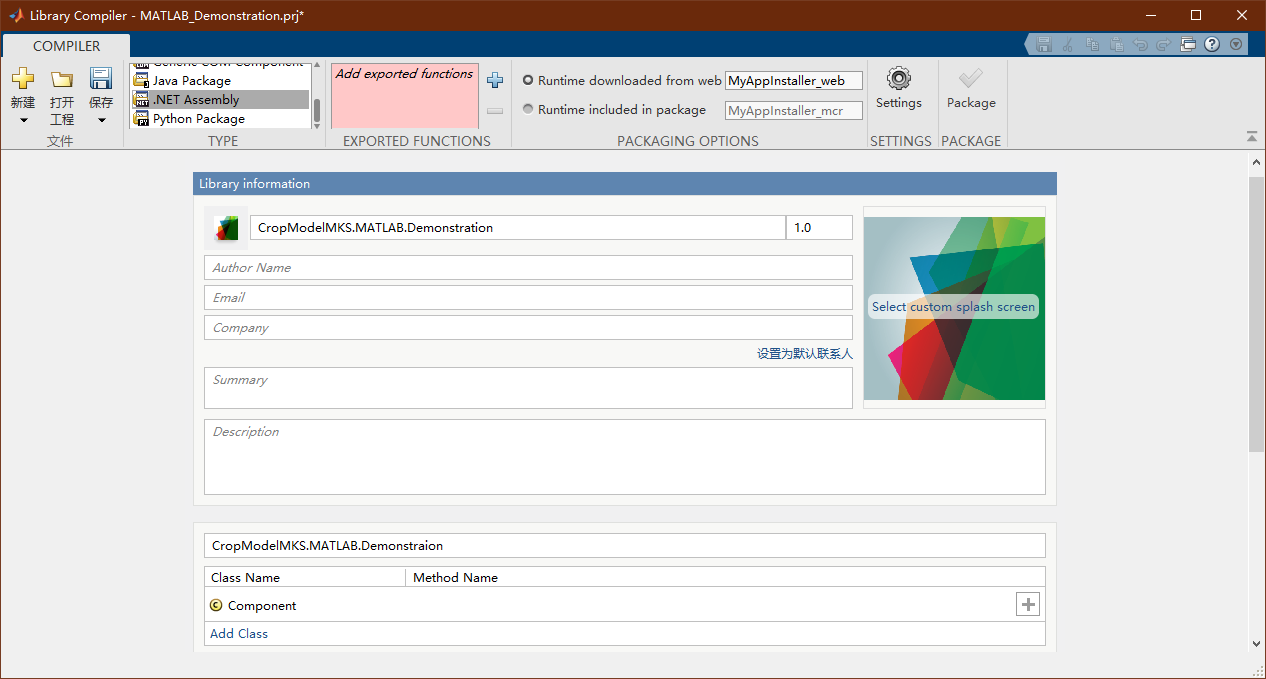
将MATLAB程序包装为CropModelMKS可以使用的组件时需要使用到MATLAB的Library Compiler，其位置如图所示。



打开该工具，选择其中的.Net Assembly，如图所示。

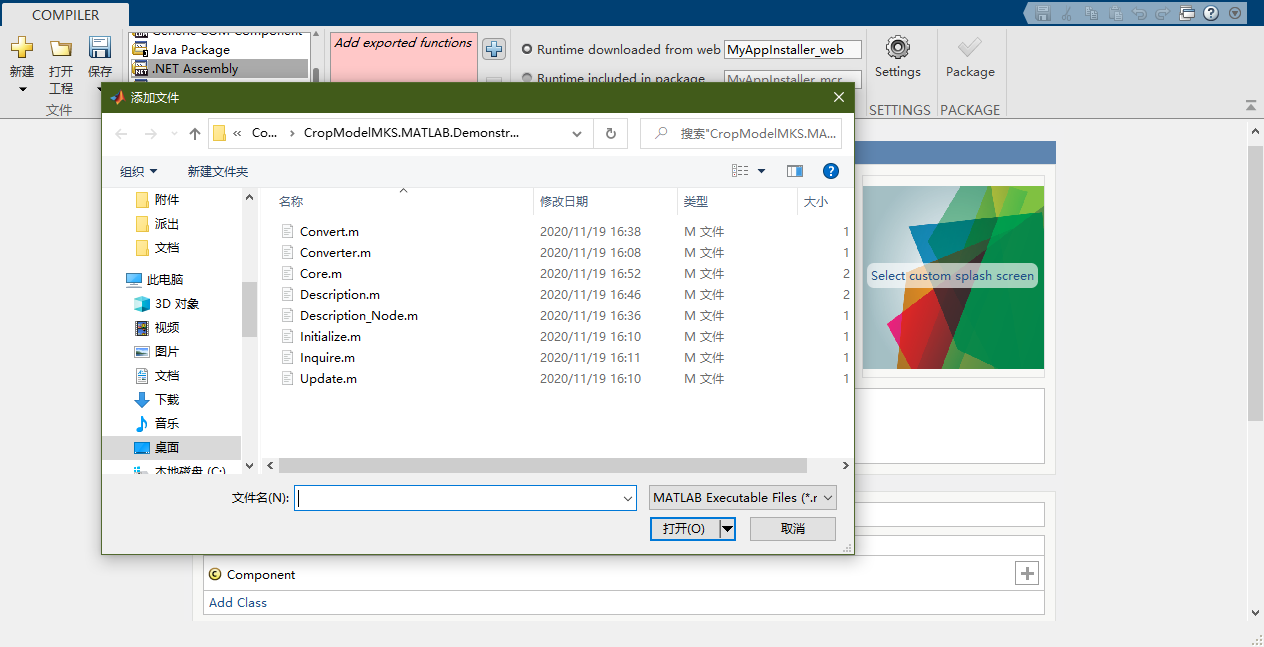


然后补充库、命名空间与类的名称信息，如图所示。



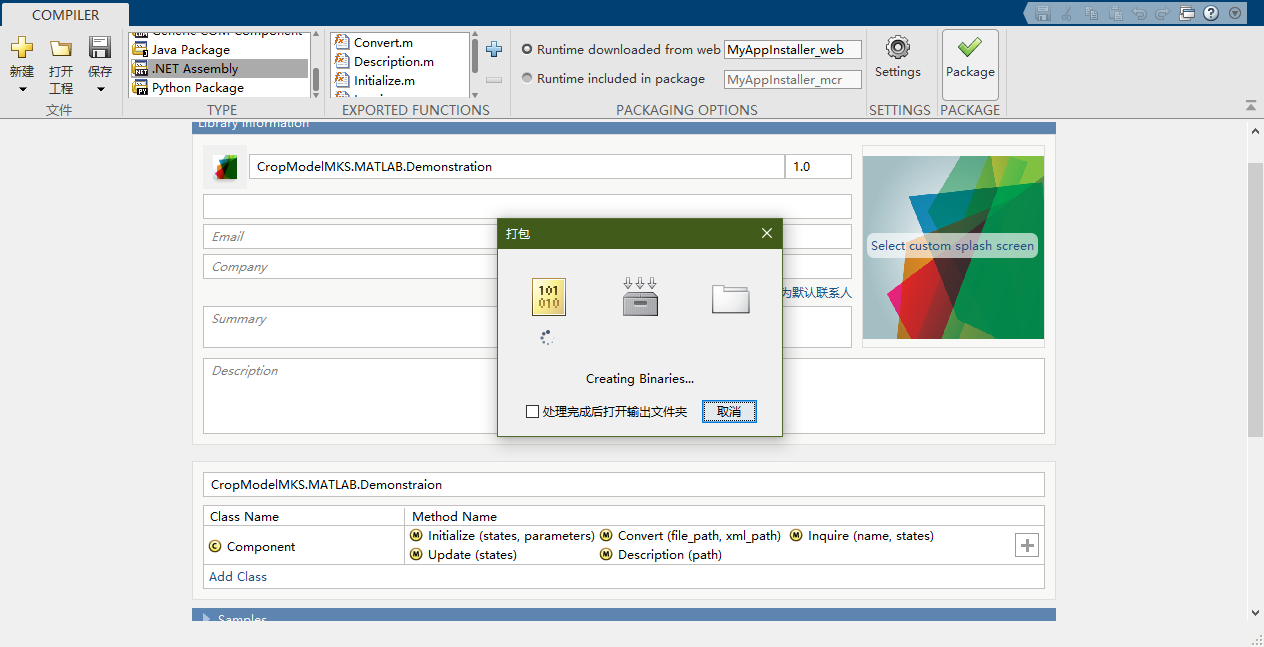
由于MATLAB中并不能将.Net Assembly通过互操作转换为COM组件，因此不能通过ProgID生成其中的类。本平台将.Net Assembly的名称作为ProgID。

然后点击蓝色加号，向工程中添加函数。如图所示。



需要向其中添加Update、Initialize、Inquire、Description与Convert。

然后点击Package按钮，开始生成动态链接库文件。点击按钮后的情况如图所示。



### Python

Python的底层库在开发时请注意包含win32com模块。

Python底层库的核心为一个Component类，类的主要结构为

××.py:

import win32com.client

import win32com.server

import pythoncom

class Component:

# Configurate the methods that could be invoked by the client.

\_public\_methods\_ = ["Initialize", "Update", "Inquire", "Convert", "Description"]

# Configurate the ProgID of this component.

\_reg\_progid\_ ="Python\_Test\_Environment"

# Create the GUID for this component.

\_reg\_clsid\_ = pythoncom.CreateGuid()

# The body of the component

def \_\_init\_\_(self):

return

def Initialize(self, states, parameters):

# This function demonstrates how to initialize the component.

# Firstly, convert the states and parameters object into the type of Dispatch

# when the two variables are passed into this function, their type is PyDispatch

# This type does not support to call the functions directly by “.”

istates = win32com.client.Dispatch(states)

iparameters = win32com.client.Dispatch(parameters)

# Then, inquire the states to be simulated by this component from states

# “climate\_states\_single” for example

result = states.Inquire("climate\_states\_single")

# if this variable has not been simulated before,

# the return value of the Inquire function should be None

# This condition indicates that you need to initialize this state from the Parameter

# And you need to write the value to the State

if result is None:

self.states.single = parameters.Inquire("climate\_states\_single")

states.Write("climate\_states\_single", self.states.single)

# When the states are finished, could then the parameters be initialized from the Parameter

# Here, the “climate\_parameters\_single” as an example

self.parameters.single = parameters.Inquire("climate\_parameters\_single")

# Initialize function finishes

return

def Description(self, path):

return

def Convert(self, file\_path, xml\_path):

return

def Inquire(self, name, states):

# This function demonstrated how to provide variables for other components

# value named “climate\_internal\_single” and array named “climate\_internal\_array” as examples

# If the variable is not supported by the component, please send None to State

istates = win32com.client.Dispatch(states)

if(name == "climate\_internal\_single"):

istates.Receive(0.12)

return

elif(name == "climate\_internal\_array"):

istates.Receive([0.3, 0.6,0.9,1.2,1.5,1.8,2.1,2.4,2.7,3.0])

return

else:

istates.Receive(None)

return

# Inquire function finishes.

def Update(self, states):

# This function demonstrates how to inquire states and variables that are not

states.Write("climate\_states\_single", 1.8)

states.Write("climate\_states\_array", [2.1, 2.2,2.3,2.4,2.5,\

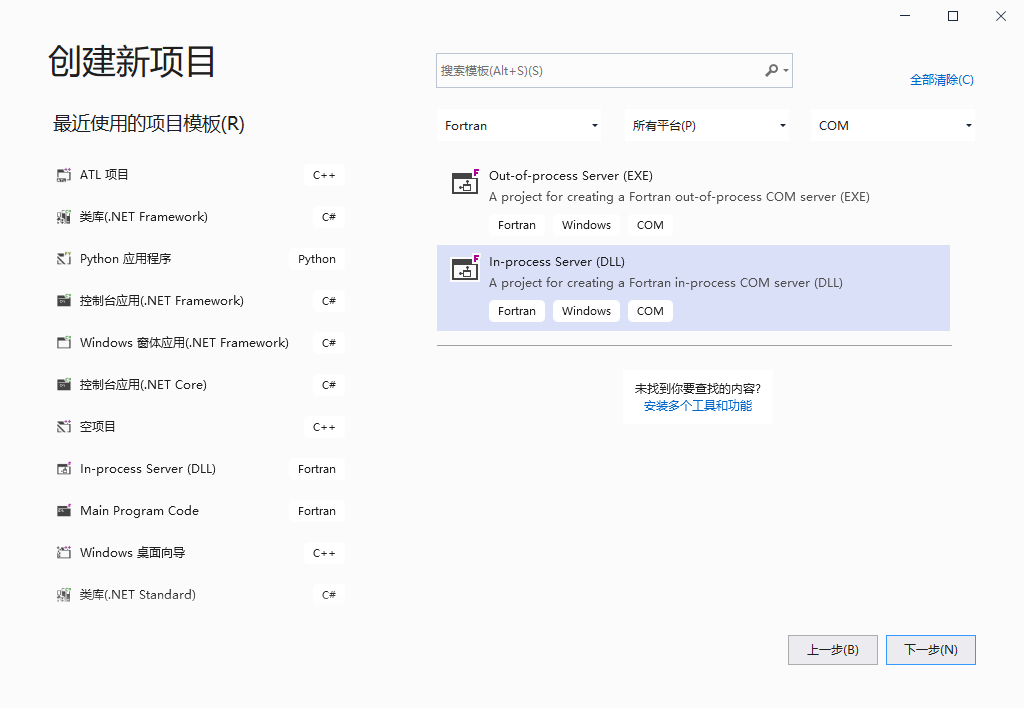
2.6,2.7,2.8,2.9,3.0])

return

### Fortran

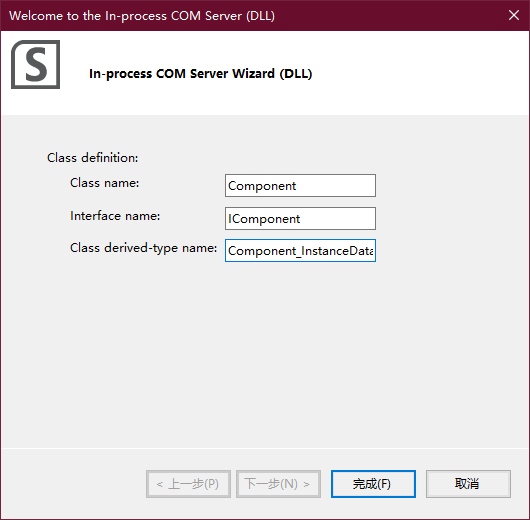
本文所介绍的Fortran库开发过程仅针对于Intel Visual Fortran编译器。在开发过程中使用到了IFWIN、IFWINTY与IFAUTO等Intel Visual Fortran所独有的模块，这些模块在其他Fortran编译器中是否有对应的实现尚不明确，使用时请谨慎。

打开Visual Studio 2019，选择创建新项目。



编程语言选择Fortran，项目类型选择COM，再选择其中的In-process Server。

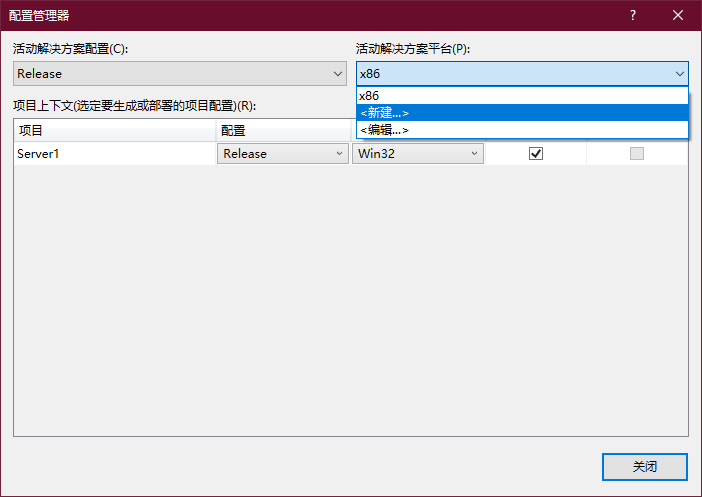
创建解决方案前，可以对类进行命名，此处仍然建议均改为Component。



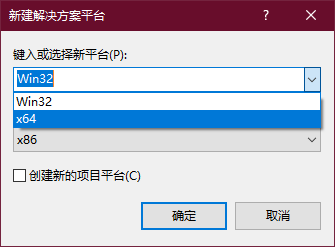
创建好解决方案后，同样需要更改项目配置。



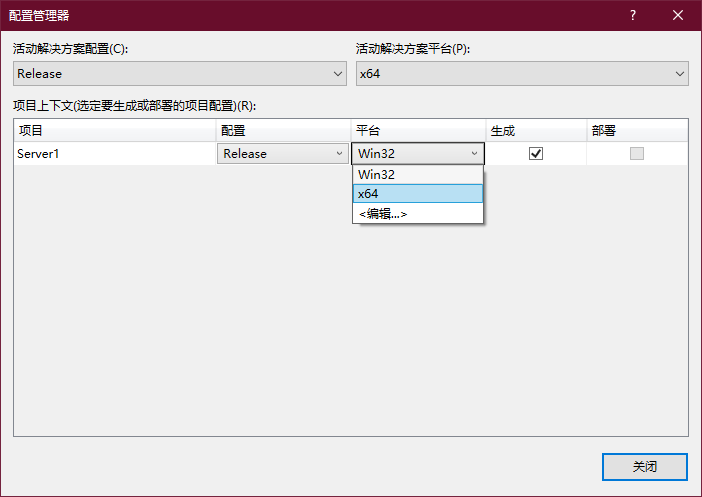
首先将，Debug改为Release，但在配置平台时，由于IVF并没有提供默认的x64平台，需要手动进行配置。如图所示。



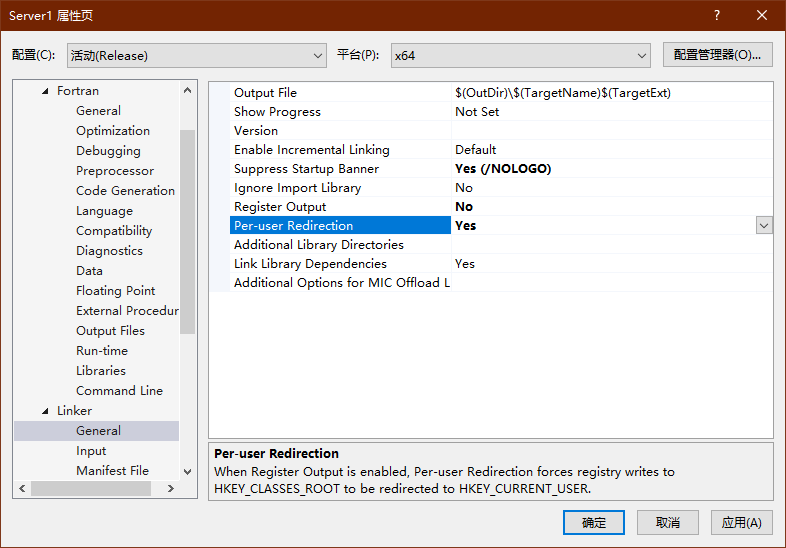
点击新建按钮，出现如下对话框。



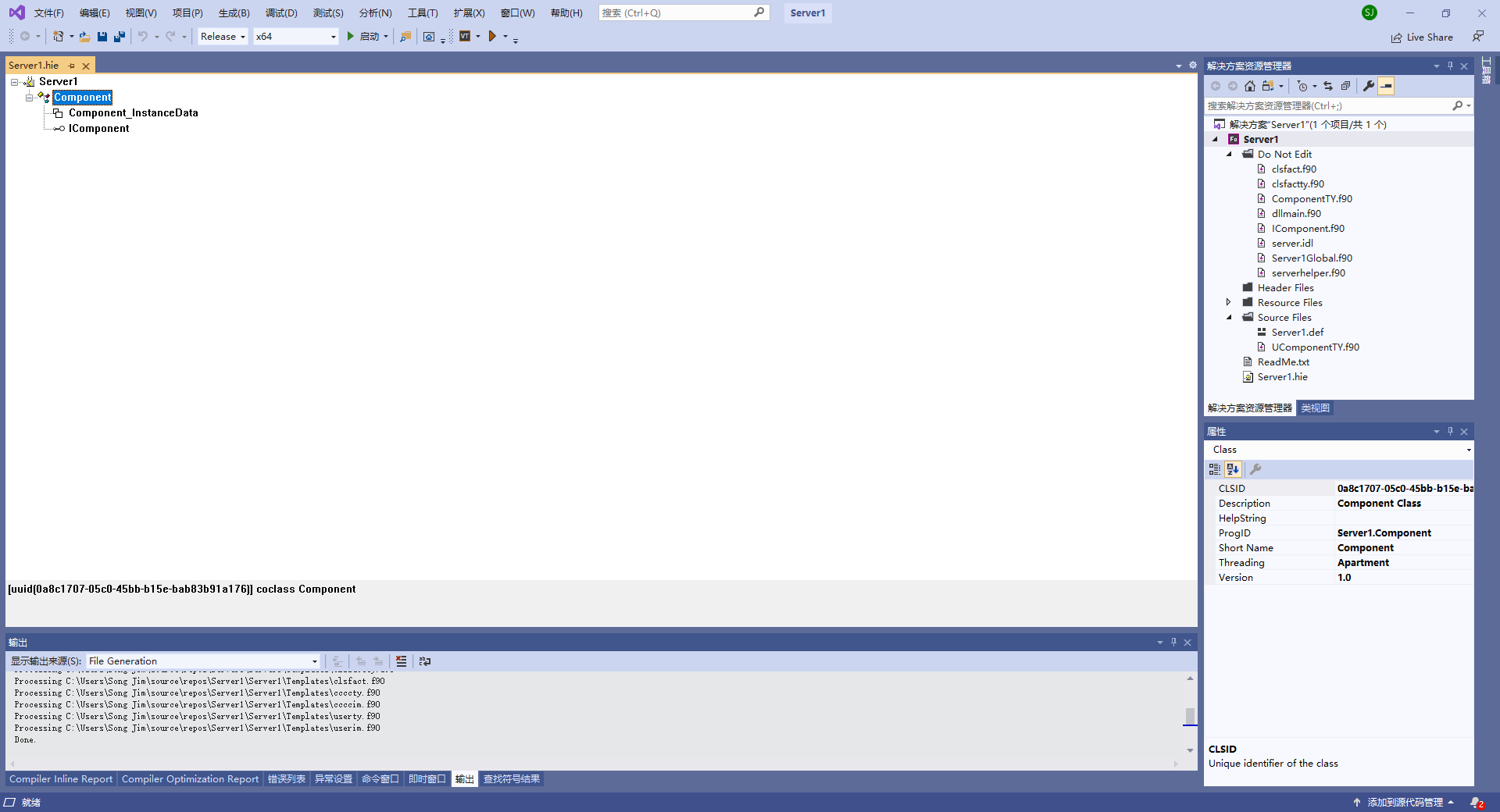
在选择x64平台，点击确定，完成x64平台配置的创建。然后将Server1项目的平台更改为x64，如图所示。



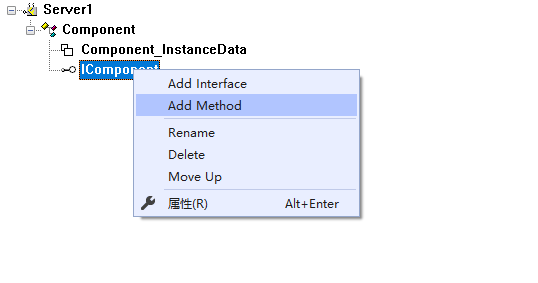
然后需要如C++一样，将注册输出改为“否”，将逐用户重定向改为“是”。



可以在Component的属性页中对ProgID进行修改，如图所示。



右键点击IComponent，弹出菜单，如图所示。

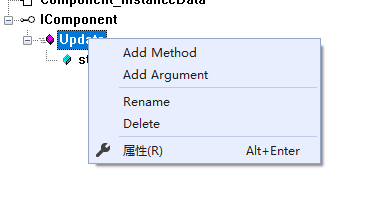


可以通过其中的Add Method添加函数。

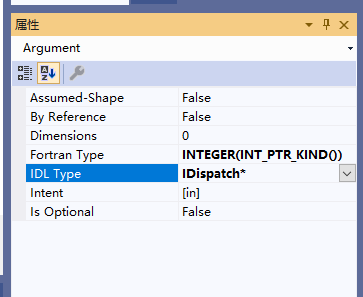
需要添加的函数有Update、Initialize、Inquire、Description与Convert。

Update函数添加方法如下：

点击上图菜单中的Add Method，然后将出现的Method改为Update，再右键点击Update，出现菜单，再点击菜单中的Add Argument，点击菜单中的Add Argument之后，再将出现的Argument改为states。如图所示。



点击states，将属性页中的Fortran Type改为INTEGER(INT\_PTR\_KIND())，将IDL Type改为IDispatch\*，如图所示。



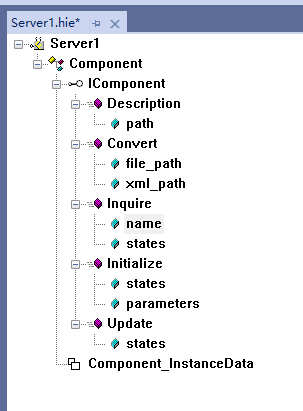
然后按照此方法依此添加Initialize、Inquire、Description和Convert方法。

添加Initialize 方法时，需要向其添加Argument：states和parameters，这二者的Fortran Type均为INTRGER(INT\_PTR\_KIND())，IDL Type均为IDispatch\*。

添加Inquire方法时，需要向其添加Argument：name与states。name的Fortran Type为CHARACTER(\*)，IDL Type已经限定为BSTR，无需更改。states的Fortran Type为INTRGER(INT\_PTR\_KIND())，IDL Type为IDispatch\*。

添加Description方法时，需要添加Argument：path，path的Fortran Type为CHARACTER(\*)。

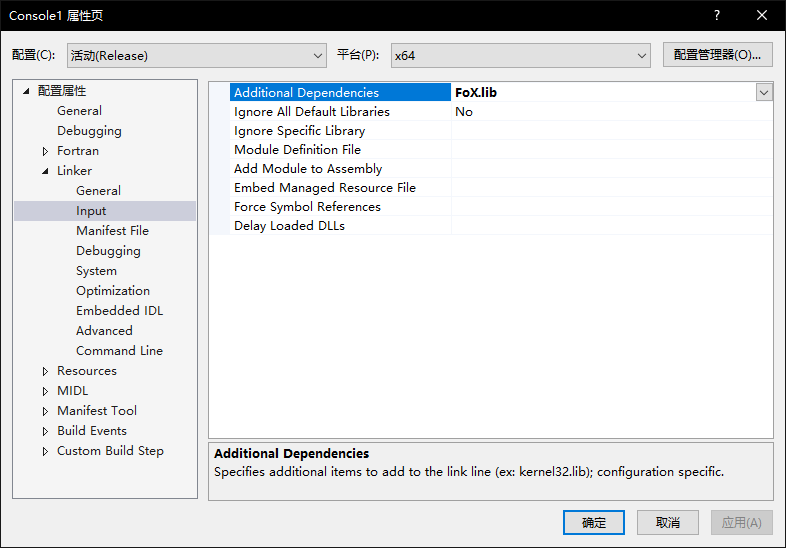
添加Convert方法时，需要添加Argument：file\_path与xml\_path，二者的Fortran Type均为CHARACTER(\*)。最终效果如图所示。



更改完毕后，通过ctrl+s保存对于Server1.hie的更改，编译器根据Server1.hie的内容自动更改相应的Fortran文件。同时，这也意味着，每次更改.hie文件时，之前对于项目中Fortran文件的更改均不会被保存。

Fortran生成XML文件采用FoX库。作者提供已经编译好的静态库（FoX.lib）和接口文件(fox\_dom.mod)。在使用时，需要将这两个文件复制到解决方案的文件夹中，然后在项目中添加对静态库的引用。过程如图所示。

右键点击项目，在菜单中选择属性。



然后在Linker-Input-Add Dependencies中输入FoX.lib。点击确定，完成对静态库的引用。

然后在项目中通过use FoX\_dom即可使用FoX库。

UI××.f90

!

! UI××.f90 - This file contains the implementation of the

! I×× methods

!

! I××\_Inquire

function I××\_Inquire( ObjectData ,&name,&states) result (hresult)

use ××\_Types

use Functions\_Module

implicit none

type(××\_InstanceData) ObjectData

!dec$ attributes reference :: ObjectData

CHARACTER(\*), intent(in) :: name

INTEGER(INT\_PTR\_KIND()), intent(in) :: states

integer(long) hresult

! <B4> DO NOT REMOVE THIS LINE

! TODO: Add implementation

integer status, i

real(kind = 8) :: single

real(kind = 8), dimension(:), allocatable :: array

if(BKDRHash(trim(name)) == BKDRHash("climate\_internal\_single")) then

status = Receive(Wrap(0.5), states)

else if (name == "climate\_internal\_array") then

allocate(array(10))

do i = 1, 10

array(i) = i \* 0.1

end do

status = Receive(Wrap(array), states)

deallocate(array)

endif

hresult = S\_OK

! <E4> DO NOT REMOVE THIS LINE

end function

! I××\_Initialize

function I××\_Initialize( ObjectData ,&states,&parameters) result (hresult)

use ××\_Types

use Functions\_Module

implicit none

type(××\_InstanceData) ObjectData

!dec$ attributes reference :: ObjectData

INTEGER(INT\_PTR\_KIND()), intent(in) :: states

INTEGER(INT\_PTR\_KIND()), intent(in) :: parameters

integer(long) hresult

! <B5> DO NOT REMOVE THIS LINE

! TODO: Add implementation

real(kind = 8) :: climate\_parameters\_single, climate\_states\_single

real(kind = 8), dimension(:), allocatable :: climate\_parameters\_array, climate\_states\_array

integer status

!the module IFWIN, IFWINTY, IFAUTO should be used

!they would dismish once the hie file is saved

!please add them after the use ××\_Types if they dismish

print\*, "Fortran Written Climate Component Initialize function entered!"

! Example of Inquireing data from states interface

status = Data\_Inquire("climate\_states\_single", states, climate\_states\_single)

if(status .EQ. -1) then

status = Data\_Inquire("climate\_states\_single", parameters, climate\_states\_single)

status = Push("climate\_states\_single", Wrap(climate\_states\_single), states)

end if

status = Data\_Inquire("climate\_states\_array", states, climate\_states\_array)

if(status .EQ. -1) then

status = Data\_Inquire("climate\_states\_array", parameters, climate\_states\_array)

status = Push("climate\_states\_array", Wrap(climate\_states\_array), states)

end if

! Example of Inquireing data from parameters interface

status = Data\_Inquire("climate\_parameters\_single", parameters, climate\_parameters\_single)

status = Data\_Inquire("climate\_parameters\_array", parameters, climate\_parameters\_array)

print\*, climate\_states\_single

print\*, climate\_states\_array

print\*, "Fortran Written Climate Component Initialized !"

deallocate(climate\_states\_array)

deallocate(climate\_parameters\_array)

hresult = S\_OK

! <E5> DO NOT REMOVE THIS LINE

end function

! I××\_Convert

function I××\_Convert( ObjectData ,&

file\_path,&

xml\_path) result (hresult)

use ××\_Types

implicit none

type(××\_InstanceData) ObjectData

!dec$ attributes reference :: ObjectData

CHARACTER(\*), intent(in) :: file\_path

CHARACTER(\*), intent(in) :: xml\_path

integer(long) hresult

! <B6> DO NOT REMOVE THIS LINE

! TODO: Add implementation

!Check the input of the file\_path,

!Change it to the open file functions when real usage

print\*, file\_path

!Chen the input of the xml\_path

!Change it to the open file functions when real usage

print\*, xml\_path

hresult = S\_OK

! <E6> DO NOT REMOVE THIS LINE

end function

! I××\_Description

function I××\_Description( ObjectData ,&

path) result (hresult)

use ××\_Types

implicit none

type(××\_InstanceData) ObjectData

!dec$ attributes reference :: ObjectData

CHARACTER(\*), intent(in) :: path

integer(long) hresult

! <B3> DO NOT REMOVE THIS LINE

! TODO: Add implementation

!Check the correctness of the input of this fucntion

!Change it to the open file functions when real usage

print\*, path

hresult = S\_OK

! <E3> DO NOT REMOVE THIS LINE

end function

! I××\_Update

function I××\_Update( ObjectData ,&

states) result (hresult)

use ××\_Types

use Functions\_Module

implicit none

type(××\_InstanceData) ObjectData

!dec$ attributes reference :: ObjectData

INTEGER(INT\_PTR\_KIND()), intent(in) :: states

integer(long) hresult

! <B2> DO NOT REMOVE THIS LINE

! TODO: Add implementation

integer(4) status, i

real(kind = 8) :: plant\_internal\_single, environment\_internal\_single, management\_internal\_single, single

real(kind = 8),dimension(:), allocatable :: plant\_internal\_array, environment\_internal\_array, management\_internal\_array, array

!the module IFWIN, IFWINTY, IFAUTO should be used

!they would dismish once the hie file is saved

!please add them after the use ××\_Types if they dismish

!The Inquire functions are stored in the file Inquire.f90

print\*, "Fortran Written Climate Component Update Function Entered!"

status = Data\_Inquire("plant\_internal\_single", states, plant\_internal\_single)

print\*, plant\_internal\_single

status = Data\_Inquire("environment\_internal\_single", states, environment\_internal\_single)

print\*, environment\_internal\_single

status = Data\_Inquire("management\_internal\_single", states, management\_internal\_single)

print\*, management\_internal\_single

status = Data\_Inquire("plant\_internal\_array", states, plant\_internal\_array)

print\*, plant\_internal\_array

status = Data\_Inquire("environment\_internal\_array", states, environment\_internal\_array)

print\*, environment\_internal\_array

status = Data\_Inquire("management\_internal\_array", states, management\_internal\_array)

print\*, management\_internal\_array

deallocate(plant\_internal\_array, environment\_internal\_array, management\_internal\_array)

single = 0.6

allocate(array(10))

do i = 1, 10

array(i) = 0.3

end do

status = Push("climate\_states\_single",Wrap(single), states)

status = Push("climate\_states\_array",Wrap(array), states)

deallocate(array)

print\*, "Fortran Written Climate Component Update Function Finished!"

hresult = S\_OK

! status = SafeArrayAccessData($Return%vu%ptr\_val, LOC(mydata))

! <E2> DO NOT REMOVE THIS LINE

end function

# 演示组件原理简介

## CropModelMKS.Climate.Default

## CropModelMKS.Plant.None

## Environment.WaterBalance.Example

## Management.Schedule.Example

## Plant.WaterDriven.Example