Python Programming Guide

GstarCAD 2027

Table of Contents

1. Introduction

GstarCAD continuously enhance the integrity and compatibility of GRX, .NET, and LISP, ensuring that users' secondary development programs run smoothly after migration, thereby reducing the cost of secondary development migration.

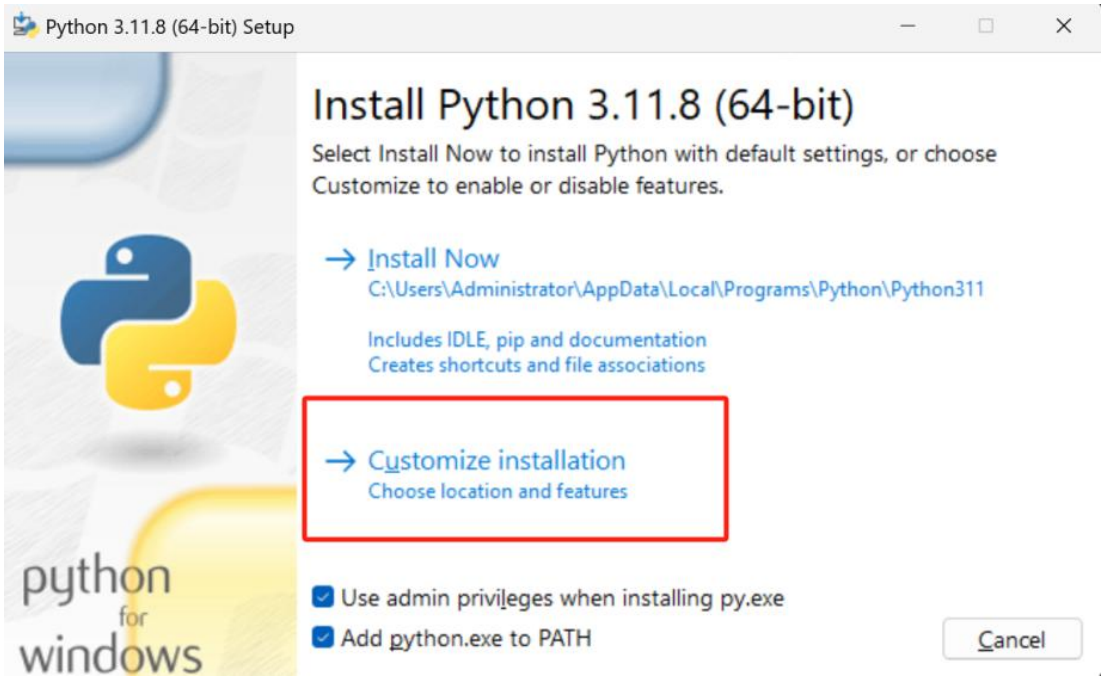
GstarCAD GRXSDK is a set of programming interfaces developed by GstarCAD that combines GRX and Python. It supports users in extending the functionality of the GstarCAD platform software using Python. With over 790 commonly used Python interfaces, we cover daily usage scenarios, such as creating custom commands, custom graphics and entities, accessing and modifying various drawing data, and implementing custom drawing and analysis algorithms. Users can leverage Python's powerful features for custom development and automation, significantly expanding GRX functionality. This script-based invocation of GRX interfaces supports cross-platform development and lowers the learning cost for new developers.

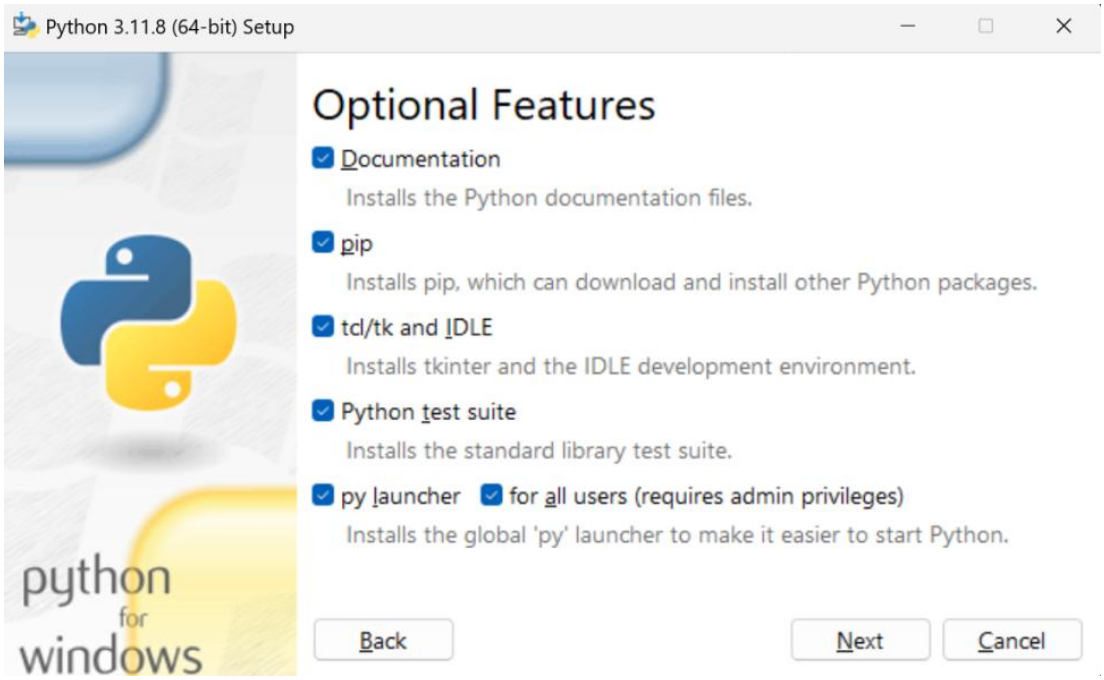
2. System Environment Requirements

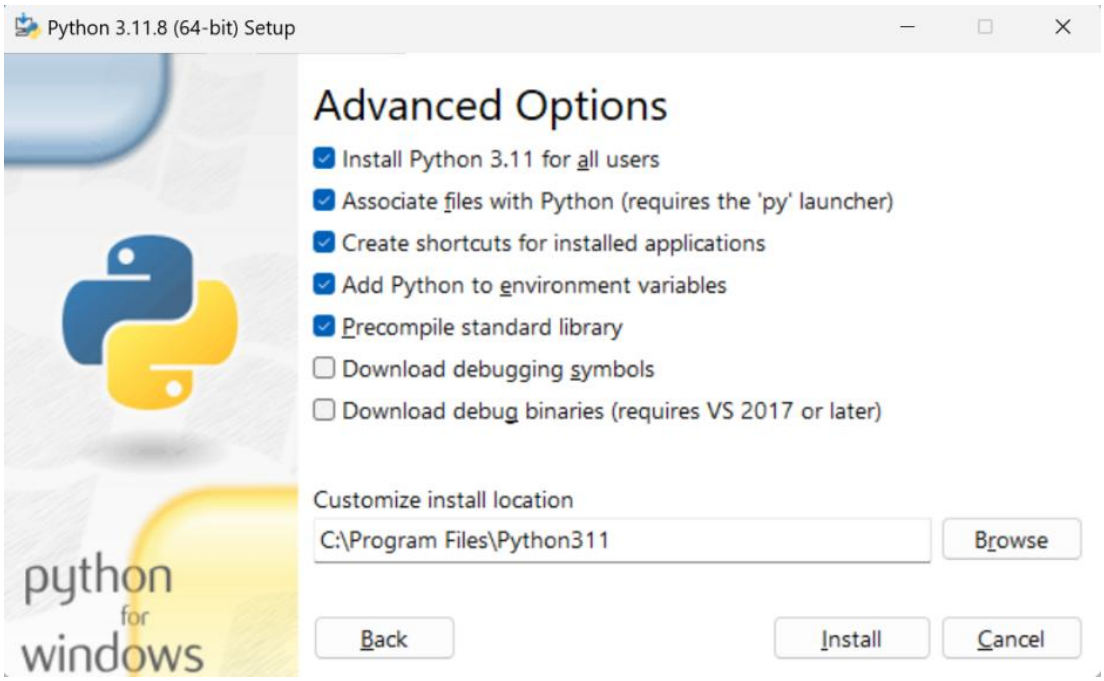
- Python 3.11.8
- Windows 10 and above

3. Install Python

Download the Python 3.11.8 installation package and install it as administrator, configure the options as below:







Then finish the installation.

4. Import Interface

Use the Import command in the Python file to import the secondary development package:

```
from pygcad.core import *
from pygcad.pygrx import *
```

pygcad.core includes Python-specific core interfaces such as the @command decorator;

pygcad.pygrx contains various types and methods corresponding to GRX interfaces.

5. Register Custom CAD Commands

During development, when a user-defined function is decorated with @command, this function is automatically registered as a command in GstarCAD, with the function name as the default command name. For example:

```
from pygcad.core.runtime import *
from pygcad.pygrx import *

@command()
def pyDrawLine():
    try:
        database = gcdbWorkingDatabase()
        (status, blockTbl) = database.getBlockTable(GcDb.OpenMode.kForRead)
        (status, record) = blockTbl.getAt(GCDB_MODEL_SPACE, GcDb.OpenMode.kForWrite)
        blockTbl.close()

        line = GcDbLine(GcGePoint3d(0, 0, 0), GcGePoint3d(100, 100, 0))
        (status, objId) = record.appendGcDbEntity(line)
        record.close()
        line.close()
    except Exception as err:
        gcedPrompt('---- [ERROR]: %s' % err)
```

The function pyDrawLine() is decorated with @command(), automatically registering it as the GCAD command PYDRAWLINE.

If you don't want to use the function name as the command name, you can specify it in the decorator function command(). The @command decorator is defined as follows:

```
def command(local_name="", global_name="", group_name="", cmd_flags=0):
```

When local_name is empty, the name of the decorated function is automatically used as local_name. If global_name is empty, it automatically uses local_name as global_name. If group_name is empty, it selects global_name as group_name. For example:

```
@command(local_name='PY_MY_CMD')
def PyFun():
    print("Custom command is PY_MY_CMD")
```

This defines a custom command: PY_MY_CMD.

6. Load in GstarCAD

For example, we used the secondary development interface to develop our own PYMKENTS command. After running the command, the program will draw a line, a circle with a red linetype, and create a layer named ASDK_MYLAYER.

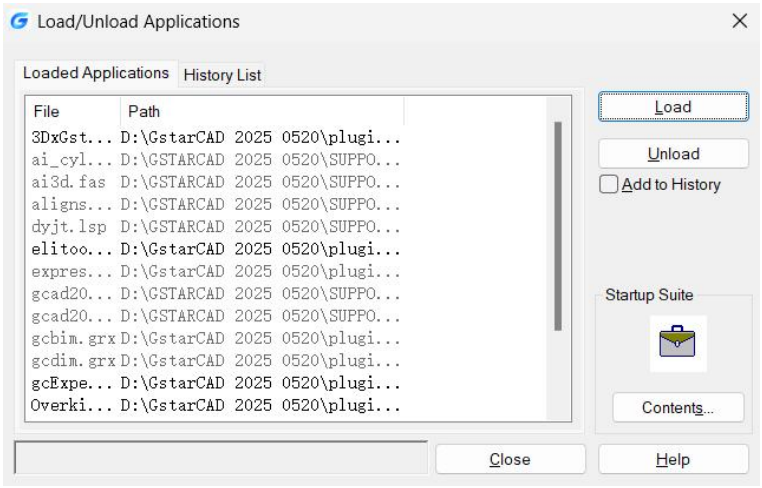
The Python file content for PYMKENTS command is:

```
from pygcad.core import *
from pygcad.pygrx import *
#import pysnooper

def createLine():
    startPt = GcGePoint3d(4.0, 2.0, 0.0)
    entPt = GcGePoint3d(10, 7.0, 0.0)
    pLine = GcDbLine(startPt, entPt)

    es, pBlockTable = gcdbHostApplicationServices(
    ).workingDatabase().getBlockTable(GcDb.kForRead)

    es, pBlockTableRecord = pBlockTable.getAt(GCDB_MODEL_SPACE, GcDb.kForWrite)
    pBlockTable.close()

    es, lineId = pBlockTableRecord.appendGcDbEntity(pLine)
    pBlockTableRecord.close()
    pLine.close()

    return lineId


def createCircle():
    center = GcGePoint3d(9.0, 3.0, 0.0)
    normal = GcGeVector3d(0.0, 0.0, 1.0)
    pCirc = GcDbCircle(center, normal, 2.0)

    es, pBlockTable = gcdbHostApplicationServices(
    ).workingDatabase().getBlockTable(GcDb.kForRead)
```

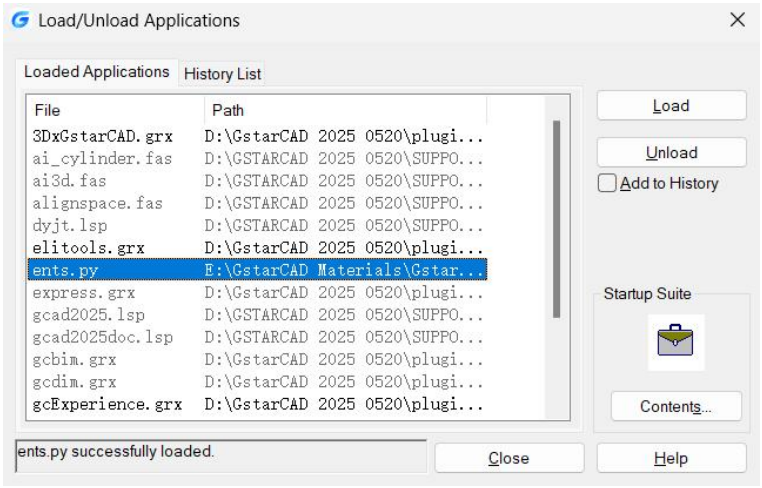
```
    es, pBlockTableRecord = pBlockTable.getAt(GCDB_MODEL_SPACE, GcDb.kForWrite)
    pBlockTable.close()

    es, circleId = pBlockTableRecord.appendGcDbEntity(pCirc)
    pBlockTableRecord.close()
    pCirc.close()

    return circleId


def createNewLayer():
    es, pLayerTable = gcdbHostApplicationServices(
    ).workingDatabase().getLayerTable(GcDb.kForWrite)

    pLayerTableRecord = GcDbLayerTableRecord()
    pLayerTableRecord.setName("ASDK_MYLAYER")

    pLayerTable.add(pLayerTableRecord)
    pLayerTable.close()
    pLayerTableRecord.close()


def createGroup(objIds=[], pGroupName=""):
    pGroup = GcDbGroup(pGroupName)

    es, pGroupDict = gcdbHostApplicationServices(
    ).workingDatabase().getGroupDictionary(GcDb.kForWrite)

    es, pGroupId = pGroupDict.setAt(pGroupName, pGroup)
    pGroupDict.close()

    for i in range(len(objIds)):
        pGroup.append(objIds[i])
```

```
    pGroup.close()


def changeColor(entId: GcDbObjectId, newColor: int):
    es, pObj = gcdbOpenObject(entId, GcDb.kForWrite)
    if pObj.isKindOf(GcDbEntity.desc()):
        pEntity = GcDbEntity.cast(pObj)
    pEntity.setColorIndex(newColor)
    pEntity.close()
    return Gcad.eOk


def runIt():
    createNewLayer()

    idArr = []
    idArr.append(createLine())
    idArr.append(createCircle())

    changeColor(idArr[-1], 1)

    createGroup(idArr, "ASDK_TEST_GROUP")


@command()
def pyMKENTS():
    runIt()
```

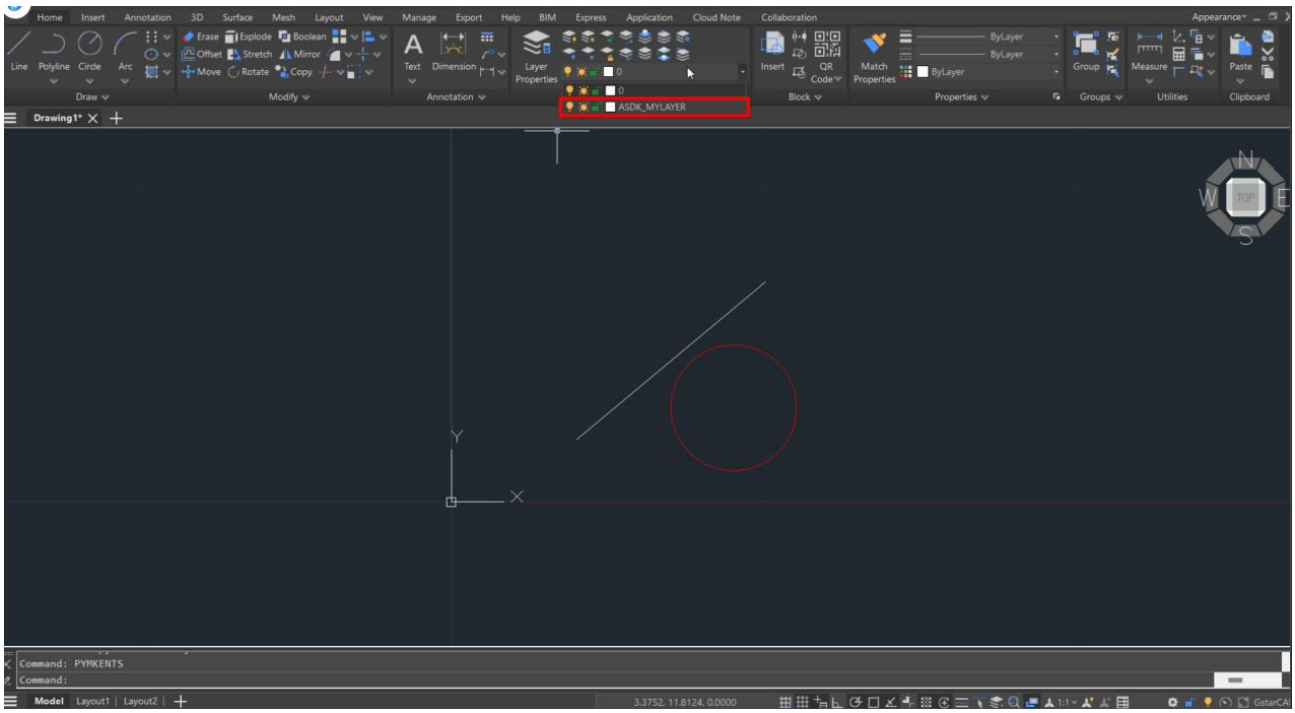
➢ Open GstarCAD 2027, enter the APPLOAD command, the Load/Unload Applications dialog box will pop up:



- Click Load button and select the ents.py file, it will register the command automatically: PYMKENTS



- Close the dialog box and enter command PYMKENTS, the program automatically runs:



GstarCAD 2027

https://www.gstarcad.net

Gstarsoft