Справка к пакету IfcOpenShell

Настоящее справочное пособие ориентировано на пользователей пакета IfcOpenShell-python.

Аннотация:

Следует понимать, что программные инструменты пакета ориентированы на упрощенный доступ к чтению файлов Ifc. В отличие от серьезных пакетов для работы с IFC типа XbimTeam, GeometryGym или даже самой IfcOpenShell но для C++, в рассматриваемом IfcOpenShell-python нет классов для каждого IFC-класса, и он не совсем предназначен для добавления новых сущностей в IFC (хоть такая возможность и есть, и мы это рассмотрим).

В составе данного пакета есть часть функционала для работы с визуализацией геометрического представления (ifcopenshell.geom) совместно с python-пакетом OCC (OpenCASCADE). На курсе эта часть подробно не рассматривается в силу возникающих прецендентов при использовании модуля, так как базовое предназначение инструмента визуализации — для работы с геометрией и её визуализации в среде OpenCASCADE. Визуализация позволяет видеть лишь геометрические объекты без атрибутики и в отношении IFC такой подход почти бесполезен, поэтому мы будем пользоваться просмотром моделей в десктопных программах. Кроме того, процедура установки Python OCC не является простой и требует нескольких больших зависимостей, что также затруднит отладку и миграцию решения впоследствии.

Помимо этого, есть огромный **недокументированный** пласт методов для создания и получения ifc-классов, о которых мы также поговорим далее.

Раздел 1 — Основные инструменты работы с IfcOpenShell

Так как python-IfcOpenShell (pyIOS далее) представляет собой простую библиотеку, в этом разделе мы рассмотрим основные классы и методы классов пакета (его API), включая также недокументированные методы.

Подраздел 1 — Обзор основных классов и их методов

Глава 1 — ifcopenshell.file

Класс «ifcopenshell.file» - это базовый класс для работа с IFC-файлами. Часть справки, посвященная ему расположена здесь. В силу того, что работать мы будем в основном, с готовыми IFC-файлами, то процедура получения файла выглядит вот так:

```
ifc_file = ifcopenshell.open(file_path)
```

Возвращаемый объект (объект ifc_file выше) — это как раз и есть класс ifcopenshell.file.
Как «основной» класс, с которого мы будем начинать работу с IFC-файлом, в нём есть методы для получения и добавления сущностей из/в файл.

Получение сущностей

Под сущностями будем понимать класс ifcopenshell.entity_instance.entity_instance (о нем см. Главу 1.2).

Есть 3 метода, позволяющих получать сущности сразу из файла. Это by_guid(guid), by_id(id) и by_type(type, include_subtypes=True).

Если цель чтения файла — получение данных по всем сущностям, то следуя иерархичности структуры Ifc-файла, удобно получать корневой IfcProject (который обязан быть в файле в единственном числе), и далее, перебирая его элементы (командой get_inverse()) получать нужные.

Если цель — получения определенных классов IFC, то удобно начать с получения списка by_type() и далее работать с этими сущностями.

Оставшийся параметр by_guid() больше свойственен «точечной» работе, например по некоему заранее подготовленному отчету проходиться по элементам файла и выбирать нужные.

Важное примечание: см. Главу 1.3 далее.

Дополнительно к трем методам выше есть ещё два метода, предоставляющие возможность выборки элементов, в составе которых встречается данный get_inverse(inst) или внутри данного (то есть тех, от которых данный элемент зависит) traverse(inst, max_levels=None). В отличие от ранних методов, они принимают в качестве аргумента результат выборки (класс ifcopenshell.entity_instance.entity_instance), но получаются только из файла, а не элемента IFC.

Глава 1.2 — ifcopenshell.entity_instance

Класс «ifcopenshell.entity_instance» - это базовый класс для всех объектов IFC. Он предоставляет методы для работы с отдельными объектами из группы или файла. У данного класса есть всего 5 документированных свойств (методов), направленных на получение информации из файла. Также есть ряд статических методов (типа walk, wrap_value), о которых мы пока не будем говорить. Часть справки, посвященная теме, здесь.

Итак, объект данного класса получается только путем выборки элемента из файла (см. Главу 1.1 выше). Пройдемся по доступным свойствам:

- attribute_name(attr_idx) получение имени атрибута объекта, по внутреннему индексу;

- attribute_type(attr) получение типа данных атрибута объекта, по внутреннему индексу;

- get_info(include_identifier=True, recursive=False, return_type=<class 'dict'>, ignore=()) получение свойств элемента в виде словаря. У метода есть ряд параметров:

-- include_identifier: boolean – включать или не включать в состав результата нумератор строки (#24= …);

-- recursive (bool) – включать ли в результат свойства зависимых элементов от данного;

-- return_type (dict|list|other): тип данных, чем будем представлен результат (словарь **«dict»** или список **«list»**);

-- ignore (set|list): список наименований атрибутов, которые не нужно включать в состав результата;

- id() - численный идентификатор строки файла (он же, так называемый STEP идентификатор);

- is_a(*args) — получение IFC-класса для объекта (функция без аргументов) или проверка, соответствует ли класс объекта заданному в аргументе классу (подача как аргумента строчного класса);

Глава 1.3 — ifcopenshell.guid

Данный класс имеет вспомогательное назначение, обеспечивая работу с объектными идентификаторами. В силу того, что по спецификации в IFC уникальный идентификатор (фиксируемый как правило в атрибуте GlobalId) — это так называемый UUID, несовместимый с GUID, необходимо иметь в виду эту несовместимость при подаче, например, функции класса ifcopenshell.file.by_guid() значения GUID.

Переход от GUID к UUID осуществляется по следующему сценарию:

```
def convert_uuid_to_guid (guid_input):
    expanded = uuid.UUID('{' + str(guid_input) + '}')
    return ifcopenshell.guid.compress(expanded.hex)
```

Подраздел 2 — О создании элементов

В данном подразделе мы создадим свой IFC-файл примитивного строения из двух элементов (тороидной стены и конусовидной кровли). Цель данного раздела — познакомить читателя с иерархией ifc-классов и логике связи геометрии и свойств с объектом внутри файла.

Мы ещё раз акцентируем внимание, что библиотека pyIOS не предназначена для создания ifc-объектов и данная демонстрация носит сугубо ознакомительный характер. Один из главных «камне преткновения» - это отсутствие документации по методам создания (так как понимание спецификации IFC не всегда однозначное).

Как и в САПР (для создания проектов) информационных моделей), для создания ifc-файла нам понадобится «шаблон», то есть минимальный набор классов (скелет файла) для работы. Опустим вариант составления и его вручную, и обратимся к вспомогательной функции `mguu_cource_tools.create_ifc_by_template()`.

В главе 1.1 среди методов класса ifcopenshell.file мы не упоминали про опции создания новых объектов. Базово, есть 1 документированный метод — create_entity(), но помимо него есть недокументированные методы вида:

ifc_file.createIfcDirection(***, ***, ***)

где create – это формальная команда/часть команды на создание объекта, а IfcDirection – нужный IFC класс, аргументы которого берутся согласно спецификации.

Более подробно процесс создания своего IFC-файла рассмотрен в видеоролике ***, поэтому здесь мы не будем комментировать процесс, оговорив лишь что все действия в видеоролике вторят спецификации IFC.

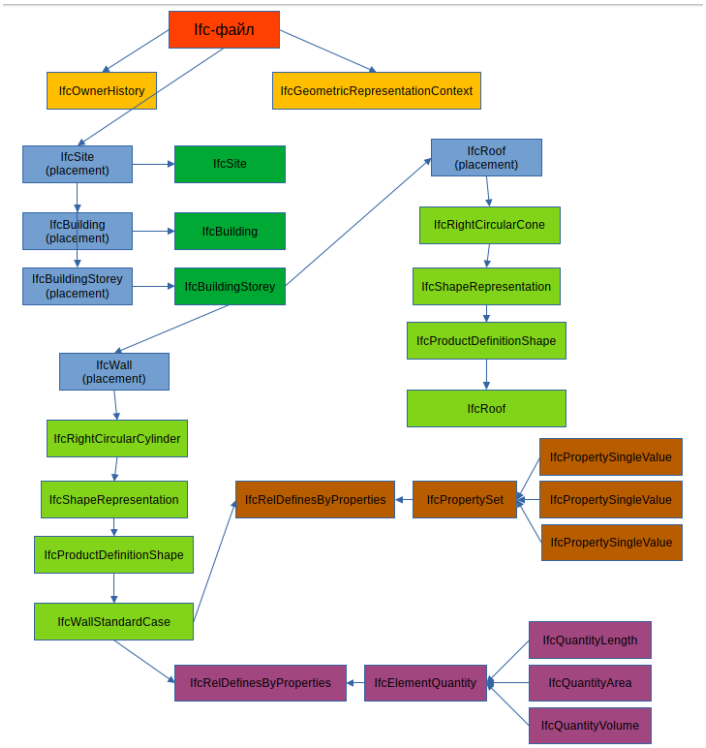

Схема иерархии классов в создаваемом файле

Подраздел 3 — О методах работы с объектами

Ранее в подразделе 2 мы коснулись недокументированных методов по созданию ifc-сущностей. Далее по курсу мы больше не будем затрагивать тему «создания элементов», но вот возможность вызова недокументированных свойств Ifc-объектов нами будет активно использоваться.

По большому счету, объект класса ifcopenshell.entity_instance будет обладать всеми свойствами обычного IFC объекта, и эти поля будут запрашиваться согласно спецификации элемента.

Например, для IfcWindow это следующие данные (впрочем, актуальные и для любого IFC-элемента):

Attribute inheritance

#	Attribute	Type	Cardinality	Description
IfcRoot				
1	GlobalId	IfcGloballyUniqueId		globally ur
2	OwnerHistory	IfcOwnerHistory	?	ssignment of the informati NOTE only the last mo IFC4 CHANGE The att
3	Name	IfcL	?	Optional name for use by th
4	Description	IfcText	?	Optional description, provid
IfcObjectDefinition				

В силу того, что наша основная работа будет связана со свойствами или (в меньшей степени), с геометрией то зафиксируем далее основные ifc-конструкции, которые можно запрашивать у объекта.

В частности, те же свойства можно получить и беря get_info() у ifc-сущности (entity_instance), но предлагаемые далее конструкции являются более «точечными». Вместе с тем, любой результат также является объектом класса ifcopenshell.entity_instance, что позволяет одновременно использовать и его методы.

```
#Получение объектных свойств (списка)
object_properties = window_instance.IsDefinedBy
for object_properties_group in object_properties:
  print("Группа свойств (объект): " + str(object_properties_group))
  properties_group_data = object_properties_group.RelatingPropertyDefinition
  print("Набор свойств (объект): " + str(properties_group_data))
  #Если это IfcPropertySet
  if properties_group_data.is_a("IfcPropertySet"):
    print("IfcPropertySet name = " + str(properties_group_data.Name))
    properties = properties_group_data.HasProperties
    for one_property in properties:
      #IfcPropertySingleValue
      print (str({one_property.Name : one_property.NominalValue}))
  elif properties_group_data.is_a("IfcElementQuantity"):
    print("IfcElementQuantity name = " + str(properties_group_data.Name))
    quanities = properties_group_data.Quantities
    for one_quanity in properties:
      #IfcPropertySingleValue
      print (str({one_quanity.Name : one_quanity.NominalValue}))
```

Пример получения свойств представлен на листинге выше.

Геометрия же получается немного по другой логике, во многом в зависимости от конкретного геометрического представления по спецификации (и версии самой спецификации). Так как от класса к классу, геометрия обычно разнородна, при цели работы с ней в IFC следует выгружать геометрию в определенном представлении. Например, в Renga это настраивается в меню настроек экспорта:

```
def get_geometry():
  #Пример для выгрузки Faceted Brep
  one_column_uuid = '0BPOXTQs7rGe4YktPDXqEu'
  column_instance = ifc_file.by_guid(one_column_uuid)
  print(column_instance)
  #Получение объектной геометрии (IfcProductRepresentation = 
IfcProductDefinitionShape)
  object_representation = column_instance.Representation
  print(object_representation.Name)
  for one_representation in object_representation.Representations:
    #IfcShapeRepresentation
    print(one_representation)
    for one_representation_item in one_representation.Items:
      print(one_representation_item)
      for one_sub_item in one_representation_item:
        print(one_sub_item)
        if one_sub_item.is_a("IfcRepresentationMap"):
          #print("MappingOrigin, placemnt = " + str(one_sub_item.MappingOrigin))
          #IfcShapeRepresentation
          shape = one_sub_item.MappedRepresentation
          print("MappedRepresentation = " + str(shape))
          for one_brep_item in shape.Items:
            if one_brep_item.is_a("IfcFacetedBrep"):
              object_IfcClosedShell = one_brep_item.Outer
              object_IfcConnectedFaceSet = object_IfcClosedShell.CfsFaces
              for one_face in object_IfcConnectedFaceSet:
                print(one_face.Bounds[0].Bound)
```

Выше представлена функция, получающая грани колонны (выгруженной предварительно из файла в форме Brep – Faceted Brep).