31.2. `imputil` — Import utilities¶

Deprecated since version 2.6: The `imputil` module has been removed in Python 3.0.

This module provides a very handy and useful mechanism for custom `import` hooks. Compared to the older `ihooks` module, `imputil` takes a dramatically simpler and more straight-forward approach to custom `import` functions.

class `imputil.ImportManager`([*fs_imp*])¶

Manage the import process.

`install`([*namespace*])¶
Install this ImportManager into the specified namespace.

`uninstall`()¶
Restore the previous import mechanism.

`add_suffix`(*suffix*, *importFunc*)¶
Undocumented.

class `imputil.Importer`¶

Base class for replacing standard import functions.

`import_top`(*name*)¶
Import a top-level module.

`get_code`(*parent*, *modname*, *fqname*)¶

Find and retrieve the code for the given module.

parent specifies a parent module to define a context for importing. It may be `None`, indicating no particular context for the search.

modname specifies a single module (not dotted) within the parent.

fqname specifies the fully-qualified module name. This is a (potentially) dotted name from the “root” of the module namespace down to the modname.

If there is no parent, then modname==fqname.

This method should return `None`, or a 3-tuple.

- If the module was not found, then `None` should be returned.
- The first item of the 2- or 3-tuple should be the integer 0 or 1, specifying whether the module that was found is a package or not.
- The second item is the code object for the module (it will be executed within the new module’s namespace). This item can also be a fully-loaded module object (e.g. loaded from a shared lib).
- The third item is a dictionary of name/value pairs that will be inserted into new module before the code object is executed. This is provided in case the module’s code expects certain values (such as where the module was found). When the second item is a module object, then these names/values will be inserted *after* the module has been loaded/initialized.

class `imputil.BuiltinImporter`¶

Emulate the import mechanism for built-in and frozen modules. This is a sub-class of the `Importer` class.

`get_code`(*parent*, *modname*, *fqname*)¶
Undocumented.

`imputil.py_suffix_importer`(*filename*, *finfo*, *fqname*)¶
Undocumented.

class `imputil.DynLoadSuffixImporter`([*desc*])¶

Undocumented.

`import_file`(*filename*, *finfo*, *fqname*)¶

Undocumented.

31.2.1. Examples¶

This is a re-implementation of hierarchical module import.

This code is intended to be read, not executed. However, it does work – all you need to do to enable it is “import knee”.

(The name is a pun on the clunkier predecessor of this module, “ni”.)

```
import sys, imp, __builtin__

# Replacement for __import__()
def import_hook(name, globals=None, locals=None, fromlist=None):
   parent = determine_parent(globals)
   q, tail = find_head_package(parent, name)
   m = load_tail(q, tail)
   if not fromlist:
       return q
   if hasattr(m, "__path__"):
       ensure_fromlist(m, fromlist)
   return m

def determine_parent(globals):
   if not globals or  not globals.has_key("__name__"):
       return None
   pname = globals['__name__']
   if globals.has_key("__path__"):
       parent = sys.modules[pname]
       assert globals is parent.__dict__
       return parent
   if '.' in pname:
       i = pname.rfind('.')
       pname = pname[:i]
       parent = sys.modules[pname]
       assert parent.__name__ == pname
       return parent
   return None

def find_head_package(parent, name):
   if '.' in name:
       i = name.find('.')
       head = name[:i]
       tail = name[i+1:]
   else:
       head = name
       tail = ""
   if parent:
       qname = "%s.%s" % (parent.__name__, head)
   else:
       qname = head
   q = import_module(head, qname, parent)
   if q: return q, tail
   if parent:
       qname = head
       parent = None
       q = import_module(head, qname, parent)
       if q: return q, tail
   raise ImportError("No module named " + qname)

def load_tail(q, tail):
   m = q
   while tail:
       i = tail.find('.')
       if i < 0: i = len(tail)
       head, tail = tail[:i], tail[i+1:]
       mname = "%s.%s" % (m.__name__, head)
       m = import_module(head, mname, m)
       if not m:
```

```
            raise ImportError("No module named " + mname)
    return m

def ensure_fromlist(m, fromlist, recursive=0):
    for sub in fromlist:
        if sub == "*":
            if not recursive:
                try:
                    all = m.__all__
                except AttributeError:
                    pass
                else:
                    ensure_fromlist(m, all, 1)
            continue
        if sub != "*" and not hasattr(m, sub):
            subname = "%s.%s" % (m.__name__, sub)
            submod = import_module(sub, subname, m)
            if not submod:
                raise ImportError("No module named " + subname)

def import_module(partname, fqname, parent):
    try:
        return sys.modules[fqname]
    except KeyError:
        pass
    try:
        fp, pathname, stuff = imp.find_module(partname,
                                              parent and parent.__path__)
    except ImportError:
        return None
    try:
        m = imp.load_module(fqname, fp, pathname, stuff)
    finally:
        if fp: fp.close()
    if parent:
        setattr(parent, partname, m)
    return m


# Replacement for reload()
def reload_hook(module):
    name = module.__name__
    if '.' not in name:
        return import_module(name, name, None)
    i = name.rfind('.')
    pname = name[:i]
    parent = sys.modules[pname]
    return import_module(name[i+1:], name, parent)


# Save the original hooks
original_import = __builtin__.__import__
original_reload = __builtin__.reload

# Now install our hooks
__builtin__.__import__ = import_hook
__builtin__.reload = reload_hook
```

Also see the `importers` module (which can be found in `Demo/imputil/` in the Python source distribution) for additional examples.

This Page

- Show Source

Navigation