Connectivity in Maple 2023

OpenMaple for Python

- The new OpenMaple for Python is an interface for the Python programming language that allows you to access Maple algorithms and data structures from a Python session on the same machine.
- You can use OpenMaple for Python from any Python session or within Maple from a Python Code Edit Region.

```python
import maple
import numpy
# Define an array of time points using numpy
timepoints = numpy.array([0, 0.25, 0.5, 0.75, 1])
# We wish to use these Maple symbols
x,y,abs,D,diff,dsolve,numeric = maple.symbols('x,y,abs,D,diff,dsolve,numeric')
# Define a differential equation in Python using syntax similar to Maple
dsys5 = {diff(y(x), x, x) + abs(y(x)) == 0, y(1) == -1, D(y)(0) == 1}
# Solve it using dsolve and sample the specified timepoints
dsol5 = dsolve(dsys5, numeric, output = timepoints)
```

Using Maple commands and symbols in Python

There are two ways to make use of Maple commands and symbols in the OpenMaple API: using **maple.symbols** or **maple.namespace.**

- **maple.symbols** lets you specify a list of symbols as a string. You can then assign these to Python variables and use them freely.

```python
import maple
evalf, ChebyshevT, gamma = maple.symbols('evalf,ChebyshevT,gamma')
res = float( evalf( ChebyshevT( 10, gamma ) ) )
```

- **maple.namespace** lets you reference any symbol from Maple with a simple prefix.

```python
import maple
import maple.namespace as mpl
res = float( mpl.evalf( mpl.ChebyshevT( 10, mpl.gamma ) ) )
```

Conversions

OpenMaple for Python can perform automatic conversions when Python objects are used as input to Maple, allowing results computed using

Python type	**Maple type**
`dict`	`table`
`list`	`list`
`set, frozenset`	`set`
`Fractions.fraction`	`fraction`
`sympy.Basic`	`anything`

Python commandline flag

The Maple commandline script on macOS and Linux now accepts an additional flag, **`-python`**, which launches the version of Python distributed with Maple. This is automatically configured to enable OpenMaple for Python to work.
To use it simply type **`maple -python`** in a terminal window and then enter **`import maple`** to load OpenMaple for Python.

Python Code Edit Regions

- Code Edit Regions now also include Python as a supported language. Simply insert a Code Edit Region into a worksheet and from the Code Edit Region properties in the Context Panel, for **Language** select **Python**. The result is a Code Edit Region which behaves identically to a Maple Code Edit Region but uses Python syntax highlighting.

```python
import numpy
def pythonFunc(n):
    return numpy.eye(n)
```

- When this region is executed, the underlying Python session associated with the Python package will be used. To access any of the variable state from this Python session, use the Python package:

```
> mapleFunc := convert( Python:-GetVariable("pythonFunc"), maple )
```

mapleFunc := **proc**( )

local *u*;

convert("<Python object: <function pythonFunc at 0x102ec1430> >"(*seq*(*convert*(*u*, '*python*'), *u* **in** [*args*])), '*maple*')

end proc

```
> mapleFunc( 5 )
```

$$\begin{bmatrix} 1. & 0. & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. & 0. \\ 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 0. & 1. \end{bmatrix}$$

Code Generation for RESTful APIs

The new OpenAPI package provides a way to automatically create Maple packages to interface with RESTful APIs from an OpenAPI specification. OpenAPI (formerly Swagger) is a standard for describing RESTful APIs with a JSON or YAML file. An example specification is provided in the examples folder to interface with the RESTful API test site at https://jsonplaceholder.typicode.com. Passing that specification to OpenAPI:-GenerateModule will create a source file for a Maple package that can be read into Maple and used to access the JSON Placeholder HTTP API.

```
> filename := cat( kernelopts(datadir), kernelopts(dirsep),
  "example", kernelopts(dirsep), "PlaceHolder.yaml"):
> output := OpenAPI:-GenerateModule( filename );
Wrote 69460 characters to PlaceHolder.mpl
```

output := "PlaceHolder.mpl"

```
> read output;
```

PlaceHolder := **module**( ) ... **end module**

```
> exports(PlaceHolder);
ClearCookies, getDefinition, getAlbums, getAlbum, getAlbumsPhotos, getComments, getComment,
    getPhotos, getPhoto, getPosts, createPost, deletePost, getPost, patchPost, updatePost,
    getPostsComments, getTodos, getTodo, getUsers, getUser, getUsersAlbums, getUsersPosts,
    getUsersTodos
> PlaceHolder:-getPost(1, 'nopopup');
table(["id" = 1, "title" = "sunt aut facere repellat provident occaecati excepturi optio reprehenderit", "body"
    = "quia et suscipit
                          suscipit recusandae consequuntur expedita et cum
                             reprehenderit molestiae ut ut quas totam
                   nostrum rerum est autem sunt rem eveniet architecto", "userId" = 1])
```