Copilot API Documentation

& business logic code migration walkthrough.

Contents

[Introduction 2](#_Toc362820674)

[Application Structure 2](#_Toc362820675)

[API Guidelines 3](#_Toc362820676)

[Loading Copilot 4](#_Toc362820677)

[Quick 4](#_Toc362820678)

[Advanced 4](#_Toc362820679)

[Creating API Routes 5](#_Toc362820680)

[Copilot Client 6](#_Toc362820681)

[Migrating Logic to Copilot 7](#_Toc362820682)

# Introduction

(c)Technical Solutions LLC. Copilot is a restful API built to fill the role of a middle tier data access layer for TS Software and Services. The purpose of this API is to provide a secure and definite home on a middle tier for application business logic.

# Application Structure

Copilot was designed as a singleton with an object oriented mindset behind the scene. Powered by the Slim framework, several core functions are segregated into their own child object instances. These functions are: logging, database connection handing, data stream compiling, and API generation. Copilot’s main object uses these children objects to handle nearly all of its functionality. Further, Copilot’s structure allows it to be run from any directory.

The inc (includes) folder contain all of Copilot’s logic.

* \_config.php
* class.api.php
* class.copilot.php
* class.data.php
* class.db.php
* class.log.php

The class folder contains any middle tier business logic which the API may need to access and run.

# API Guidelines

Your developer should be able to hit the keyboard and get something useful back.

Some soft guidelines for designing a REST architecture.

1. *Do not use "physical" URLs. A physical URL points at something physical -- e.g., an XML file: "http://www.acme.com/inventory/product003.xml". A logical URL does not imply a physical file: "http://www.acme.com/inventory/product/003".*
   * *Sure, even with the .xml extension, the content could be dynamically generated. But it should be "humanly visible" that the URL is logical and not physical.*
2. *Queries should not return an overload of data. If needed, provide a paging mechanism. For example, a "product list" GET request should return the first n-products (e.g., the first 10), with next/prev links.*
3. *Even though the REST response can be anything, make sure it's well documented, and do not change the output format lightly (since it will break existing clients).*
   * *Remember, even if the output is human-readable, your clients aren't human users.*
   * *If the output is in XML, make sure you document it with a schema or a DTD.*
4. *Rather than letting clients construct URLs for additional actions, include the actual URLs with REST responses. For example, a "product list" request could return an ID per product, and the specification says that you should use http://www.acme.com/product/PRODUCT\_ID to get additional details. That's bad design. Rather, the response should include the actual URL with each item: http://www.acme.com/product/001263, etc.*
   * *Yes, this means that the output is larger. But it also means that you can easily direct clients to new URLs as needed, without requiring a change in client code.*
5. *GET access requests should never cause a state change. Anything that changes the server state should be a POST request (or other HTTP verbs, such as DELETE).*

(ELKSTEIN, 2008)

Further, you should note copilot’s formatting for passing filters and fields into the URL. Both of these are enclosed in delimiters and a specific symbol to specify what kind of data is between the delimiters.

|  |  |
| --- | --- |
| C:\Program Files (x86)\Zend\Apache2\htdocs\copilot\docs\imgs\url_with_filter_field.PNG | example URL |
| C:\Program Files (x86)\Zend\Apache2\htdocs\copilot\docs\imgs\filter.PNG | filters |
| C:\Program Files (x86)\Zend\Apache2\htdocs\copilot\docs\imgs\field.png | fields |

# Loading Copilot

## Quick

Using this method, your instance of Copilot will be called $\_\_CP

C:\Program Files (x86)\Zend\Apache2\htdocs\copilot\docs\imgs\load_quick.png

## Advanced

You must only run $INSTANCE->ready() after you have created the API routes.

C:\Program Files (x86)\Zend\Apache2\htdocs\copilot\docs\imgs\load.PNG

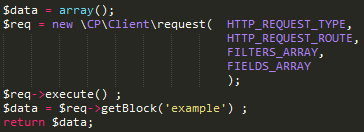
# Creating API Routes

C:\Program Files (x86)\Zend\Apache2\htdocs\copilot\docs\imgs\createroute.PNG

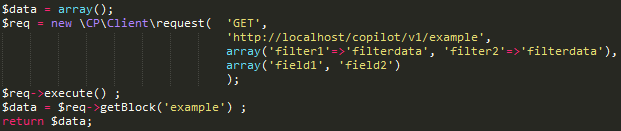
|  |  |  |
| --- | --- | --- |
| **HTTP Request Type** | : | @PARAM STRING - GET, POST, PUT, or DELETE  e.g.  **‘GET’** |
| **HTTP Request Route** | : | @PARAM STRING - The URL extension which will call this API route.  e.g.  **‘/users’** |
| **Callback Action** | : | @PARAM FUNCTION - A function which can be used as a callback.  e.g. A lambda function.  **function() use ($\_\_CP) {**  **echo “this is an API callback function!” ;**  **}** |
| **Callback Header** | : | @PARAM FUNCTION (OPTIONAL) – A function which can be used as a callback. This function call runs before the first callback. It was designed for running include code that runs in nearly all of your calls.  e.g. A lambda function. (see previous) |

# Copilot Client

Example copilot call.

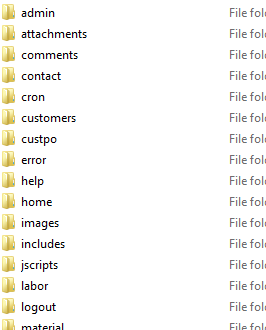
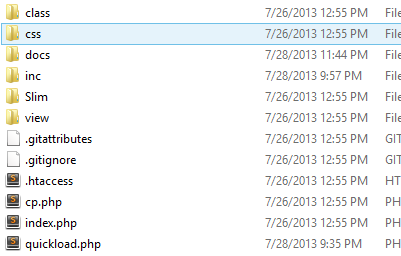


Example.



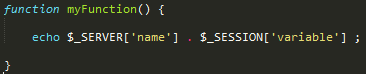
# Migrating Logic to Copilot

1. Copy code containing logic from its old location to Copilot.

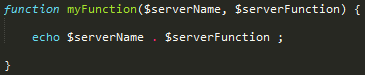


1. Edit dependencies out of copied code.

*BEFORE*



*AFTER*



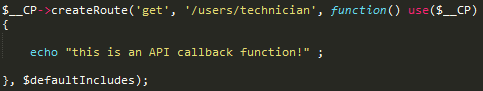
1. Include copilot in your application code. (quickload.php)

*See the section on “Loading Copilot”.*

C:\Program Files (x86)\Zend\Apache2\htdocs\copilot\docs\imgs\load_quick.png

1. Create a new route which includes your copied and edited.

*See the section on “Creating API Routes”.*



1. Edit the old code to call the Copilot API. Using the copilot client, call the new route you created in place of the old code in your application.

*See the section on “Copilot Client”.*

