

LLMs, Accessibility, MAUI, Game Development

CODE

NOV
DEZ
2024

codemag.com - THE LEADING INDEPENDENT DEVELOPER MAGAZINE - US \$ 8.95 Can \$ 11.95

CODE

30 YEARS

Simple Game Development with AI

Deep Dive into
Parallel C#

Line of Business
Apps with MAUI

Preparing Your
Applications
for Accessibility





**ARE YOU WONDERING
HOW ARTIFICIAL
INTELLIGENCE CAN
BENEFIT YOU TODAY?**

EXECUTIVE BRIEFINGS

Are you wondering how AI can help your business? Do you worry about privacy or regulatory issues stopping you from using AI to its fullest? We have the answers! Our Executive Briefings provide guidance and concrete advice that help decision makers move forward in this rapidly changing Age of Artificial Intelligence and Copilots!

We will send an expert to your office to meet with you. You will receive:

1. An overview presentation of the current state of Artificial Intelligence.
2. How to use AI in your business while ensuring privacy of your and your clients' information.
3. A sample application built on your own HR documents – allowing your employees to query those documents in English and cutting down the number of questions that you and your HR group have to answer.
4. A roadmap for future use of AI catered to what you do.

AI-SEARCHABLE KNOWLEDGEBASE AND DOCUMENTS

A great first step into the world of Generative Artificial Intelligence, Large Language Models (LLMs), and GPT is to create an AI that provides your staff or clients access to your institutional knowledge, documentation, and data through an AI-searchable knowledgebase. We can help you implement a first system in a matter of days in a fashion that is secure and individualized to each user. Your data remains yours! Answers provided by the AI are grounded in your own information and is thus correct and applicable.

COPILOTS FOR YOUR OWN APPS

Applications without Copilots are now legacy!

But fear not! We can help you build Copilot features into your applications in a secure and integrated fashion.

CONTACT US TODAY FOR A FREE CONSULTATION AND DETAILS ABOUT OUR SERVICES.

codemag.com/ai-services

832-717-4445 ext. 9 • info@codemag.com

Features

? CODE: Five Years Ago

Markus finishes up his series on the last 30 years with a look at the most recent changes in computing and what's been happening with the business of CODE.

Markus Egger

? AI with No Internet Connection

If you want to build something without giving your code to the faceless entities on the cloud before you've even run tests or put it into production, you need to build it offline. Sahil shows you how you can access this powerful new tool without the online risk.

Sahil Malik

? Exploring .NET MAUI: Data Entry Controls and Data Binding

In this third entry in a series on MAUI, Paul delves into making the pages you created in the first two articles even more functional using data entry controls and data binding.

Paul Sheriff

? First Rule of ARIA: Don't Use ARIA

As an accessibility tool, ARIA (Accessible Rich Internet Applications) can be mystifying. Ashlei explains what it is, when NOT to use it, and why.

Ashlei Lodge

? Can an LLM Make a Video Game?

ChatGPT and others of its ilk appear in every news story, every board meeting, and every online search. Can we embrace this technology and have fun with it? Jason rebuilds a game from his childhood and has a great time doing it.

Jason Murphy

? Threads, Asynchrony, Parallelism, and Concurrency in C#

Even if you have a lot of resources at your disposal, there are some rules of processing that will help you keep things efficient and running smoothly. Joydip explains some of these basic tools and shows you how to get the most from them.

Joydip Kanjilal

Departments

6 Editorial

? Advertisers Index

74 Code Compilers



US subscriptions are US \$29.99 for one year. Subscriptions outside the US pay \$50.99 USD. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa, and Discover credit cards are accepted. Back issues are available. For subscription information, send e-mail to subscriptions@codemag.com or contact Customer Service at 832-717-4445 ext. 9.

Subscribe online at www.codemag.com

CODE Component Developer Magazine (ISSN # 1547-5166) is published bimonthly by EPS Software Corporation, 6605 Cypresswood Drive, Suite 425, Spring, TX 77379 U.S.A. POSTMASTER: Send address changes to CODE Component Developer Magazine, 6605 Cypresswood Drive, Suite 425, Spring, TX 77379 U.S.A.



Creative versus Reactive

I'm writing another book, a mystery this time. Of course, it's riddled with music history (and some cool stuff about ancient maps and ship's journals from the 15th century). I've invented a situation where something believed to be lost is found and there's a chase across Germany to discover the truth (or not) of it.

I'm having a lot of fun combining musicology and history with modern technology. But there's a catch.

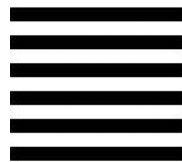
After a little bit of research, I've discovered that books in the mystery section almost always include a murder. In order to get published you have to play by the rules, so I started thinking about which character I was willing to bump off. The answer is that I don't want to kill any of them, partially because I already have two other books planned in the series and I need the expertise of the main characters in all of them. I don't want to distract from my chase through history with a murder, which, after all, should be more important than some missing or rediscovered documents, right? Now that I've done this tiny bit of research, my writing process is colored by the question: How risky is it to NOT do what everyone else is doing?

We see this dilemma all the time in software development. Something truly innovative succeeds and immediately the bandwagon is full of imitators and knock offs. Some companies even do "variations on a theme" themselves in an effort to squeeze every drop of profit and reputation out of their product. Even if you know you're first, how do you know if what you've built is the first of a great thing or just another forgotten blip in the history of software? How do you know whether the enthusiasm of the team is the pleasure of the problem-solving chase and or if it's a truly world-changing product?

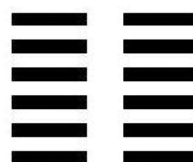
I started thinking about inventions that changed the world but that a later invention improved upon. For instance, China came up with the idea for moveable type about three centuries earlier than Gutenberg, but Gutenberg's printing press made literacy available to the masses for the first time—he didn't know about the Chinese invention, so he invented moveable type on his own. Or how about Orville and Wilbur Wright? Although they weren't the first to have their idea, their success made commercial flight possible only 11 years later. Apple's GUI and, shortly thereafter, Windows 95 made it possible for smart phones to be in every pocket just over twenty years later. Now, a whole generation can't imagine a time before portable computers and internet access.

There's obviously plenty of evidence for breaking new ground leading to success, and plenty of evidence for success coming from improving upon something that already exists. How do you know if it's too great a risk or one that the world just isn't ready for? Or even that the problem you've solved is one that you alone had?

All this hemming and hawing made me think about the creative impulse and the imitative improvements that are everywhere, occasionally obscuring the original completely, and the risks involved with being completely original and with being yet another wannabe. I remembered lessons I learned years ago from the *I Ching*, the late 9th century BCE divination text that so captivated the great philosopher Confucius (5th and 4th centuries BCE). The divinator tosses yarrow sticks to get a hexagram that reveals the answers to questions or about the person asking the questions. The two most rare hexagrams are the Creative (#1) and the Reactive (#2) (see **Figure 1**). The Creative comes up with true innovations, apparently out of nowhere, and the Reactive takes an idea that already exists and tweaks it into something new. These ancient concepts are still applicable, which is kind of cool, but not really helpful in my novel-writing conundrum. This isn't a new concept in my own life either, as my non-fiction music history book is entirely based on a study of innovation—both those that came from nowhere and those based on existing musical inventions (<https://tinyurl.com/MusicalInnovators>).



Hexagram 1



Hexagram 2

One way of invention isn't better than the other—they're just different. You can tweak an existing thing so extensively that it barely resembles the original, and you can come up with something entirely on your own that's

exactly like something someone else invented elsewhere. (Did you know that the bagpipe was invented on every single populated continent except Australia? Australians had their own drone instrument—the diggeridoo. Apparently, everyone loved a drone once upon a time, and everyone—even in Australia—found a good use for a nice animal bladder.) Anyway. I started thinking about how we're supposed to know when our innovation is truly new and will be popular or useful, and when it's merely another variation on a theme and will take only a small nibble out of an already well-munched pie.

I've only written five chapters in the first draft of my book, so I think I'll finish writing the story I want to tell without a murder. There will be more drafts. I've got more research to do about whether I'm the only non-murderous mystery writer, but I've got months to go on my first draft, so plenty of time to change my mind, if I get to the end and feel that it needs something more exciting than just solving three ancient mysteries with a wild chase across Europe.

Will you succumb to peer pressure when you're developing software? Or go your own way and hope that yours is the start of a whole new direction?

Melanie Spiller
CODE

CODE: 5 Years Ago

**Markus Egger**

megger@codemag.com

Markus, the dynamic founder of CODE Group and CODE Magazine's publisher, is a celebrated Microsoft RD (Regional Director) and multi-award-winning MVP (Most Valuable Professional). A prolific coder, he's influenced and advised top Fortune 500 companies and has been a Microsoft contractor, including on the Visual Studio team. Globally recognized as a speaker and author, Markus's expertise spans Artificial Intelligence (AI), .NET, client-side web, and cloud development, focusing on user interfaces, productivity, and maintainable systems. Away from work, he's an enthusiastic windsurfer, scuba diver, ice hockey player, golfer, and globetrotter, who loves gaming on PC or Xbox during rainy days.



dtSearch®



Instantly Search Terabytes

dtSearch's **document filters** support:

- popular file types
- emails with multilevel attachments
- a wide variety of databases
- web data

Over 25 search options including:

- efficient multithreaded search
- **easy** **multicolor** **hit-highlighting**
- forensics options like credit card search

Developers:

- SDKs for Windows, Linux, macOS
- Cross-platform APIs cover C++, Java and current .NET
- FAQs on faceted search, granular data classification, Azure, AWS and more

Visit dtSearch.com for

- hundreds of reviews and case studies
- fully-functional enterprise and developer evaluations

The Smart Choice for Text Retrieval® since 1991

dtSearch.com 1-800-IT-FINDS

Markus Egger
CODE



AI with No Internet Connection

AI, or artificial intelligence; are you bored of hearing about of it yet? Between the stock market and CEO keynotes, we can't seem to get away from it. It promises to revolutionize everything around us. We'll have robots mowing our lawns, artificial intelligence teachers teaching us on our tablet computers, etc. We've all seen those incredible demos, yet when I want to build something



Sahil Malik

www.winsmarts.com
@sahilmalik

Sahil Malik is a Microsoft MVP, INETA speaker, a .NET author, consultant, and trainer.

Sahil loves interacting with fellow geeks in real time. His talks and trainings are full of humor and practical nuggets.

His areas of expertise are cross-platform Mobile app development, Microsoft anything, and security and identity.



useful with it, it comes down to a hard choice of rote theoretical concepts that seem so far away from usable code, or a matter of calling APIs with a key and giving up all my data to some random company out there. Where's that robot mowing my lawn?

The issue is that I don't want to offer up the corpus of my data to some random service out there. Maybe it's a data security or privacy issue, or maybe I need quicker response times. Maybe it's a matter of ongoing cost, or maybe I just don't have internet connectivity. For instance, wouldn't it be nice if I could just ask questions of "my" email over a simple chat-based UI, without having to share my life history with Apple or Microsoft? What if I was taking up a new job, and the new job shared 50 pages of fine-print details with me? I have many questions. I must accept the offer within 24 hours, and I have so many questions. I wish I could just ask questions. Or what if I had lots and lots of old financial data, and I wanted to ask a simple question, like "What is this \$58.76 expense about?" and my computer had the intelligence to OCR all my receipts, and answer my questions in simple English, like "This \$58.76 receipt was for tolls on your trip to customer xyz in city def". Or maybe my lawn-mowing robot ran out of Wi-Fi range and needs to decide quickly if it's okay to mow over the rabbit? For the record, it is never okay to do that.

I've seen all these demos, yet when I sit down to solve these basic problems, all these promises by all these magnificent companies fall short.

Why can't I ask my computer such simple questions?

I'm a developer, so I set out to solve this problem. By the end of this article, I'll show you how you can build an application that works completely offline, sends no data to the cloud, and is able to answer simple questions from any source of knowledge. Sort of your own private ChatGPT.

For fun, I'll use my previous CODE Magazine article about VSCode tips (<https://www.codemag.com/Article/2409031/More-VS-Code-Tips>) and ask some simple questions about VSCode. But you can point this application to really any source of data—your company's documentation, or the health benefits of your HR department, or the state of the union, or the quarterly report of a company, or some deep research on quantum physics—and ask a simple question, like "What's an infinite well" or "What's the copay if I visit a specialist" or "Do I need a referral for a specialist in my medical plan" or "What's the chips and science act" and get legit answers.

Tell me: Would you find it useful if you had a PDF manual for your car, and could ask a simple question like, "Does the B2 service for my car involve a coolant flush?"

Let's build an application that takes in a corpus of any offline information and allows you to ask questions of it.

What You're Going to Need

To follow this article, you'll need a beefy machine. You're not going to rely on the cloud to build the model for you. You'll need a powerful local compute capability. This means either a higher-end Windows/Linux laptop, or one of the newer Macs. And yes, you'll need a GPU. AI involves a lot of calculations, and to speed things up, a lot of them are offloaded to the GPU. The difference between doing everything on the CPU vs. GPU is astronomical. For my purposes, I'll use my rusty trusted M1 Max MacBook Pro. It's a few years old, but it has enough oomph to work on thousands of pages of text, which is good enough for my needs. Hopefully, you have a similarly equipped machine, or to follow along you could just use a smaller input dataset.

Also, you'll download and use standard libraries, packages, and large language models that other companies and people have built. But when you're done with it, no data will be sent to the cloud, and your application will have the ability to be able to run completely offline. To get started, you'll need an internet connection.

Also, I'll use Python, so ensure that you have Python 3x installed.

Ollama

Ollama helps you run local large language models. There are many ways to run a large language model locally, and Ollama is one of the easiest ways to get started. Ollama is to large language models a bit like what Docker is to your code. Note that Ollama isn't the only way to run local language models, but it's one of the easiest to get started with and has a nice library of models you can pick from. There are many other choices when it comes to running language models locally. Some examples are hugging face transformers, LLaMA, Mistral, LocalLLaMA, Cerebras, etc.

The thing is, my puny little Mac isn't going to learn to understand English or Spanish on its own. It's going to need help. And large companies like Meta, Microsoft, and Google have spent billions of dollars of compute and terabytes of data to create large language models to get us started. Although I do want them to understand my corpus of data, it certainly helps that they already understand so much else. The model I downloaded is just a gigantic formula they've pre-built for me. All I have to do is call the formula with my inputs.

Picking the Right Model

You can see that Ollama already supports nearly any large language model you may be interested in here: <https://>

ollama.com/library. There are many large language models that are general purpose, such as Gemma2 from Google, or Llama3 from Meta, or Phi3 from Microsoft. Each of these models is a snapshot of the world—around when the model was created and published. Each of these have their own weaknesses and strengths, some are designed for accuracy, some are designed for speed. You can also mix and match these models. For instance, maybe my target application is very code-centric. Why should I pay the overhead for the knowledge of what kind of oil a Hyundai engine needs? I would, instead of using Llama3, use Code Llama instead. Code Llama is a model that's designed for generating and discussing code. It's built on top of Llama2. It can both generate code and natural language about code.

When picking a model, there are a few things to consider. Naturally, every large tech company has a model to offer, and everyone will tell you their model is the best. The reality is that they're all imperfect. And depending upon how you choose to run a model, you may or may not be able to pick the model you like.

When picking a model, typically the model will be available in model or parameter sizes. For example, Llama3.1 is available in 8B, 70B, or 405B parameter sizes. That's B for Billion.

When we speak of large language models, model size refers to the number of parameters in the model. Parameters are the internal variables that the model uses to make predictions or generate text.

Think of parameters as the possible number of inputs to a formula. The more parameters a model has, the more complex and nuanced its behavior can be. However, increasing the number of parameters also increases the risk of overfitting. Overfitting is a common problem in machine learning, including large language models. It occurs when a model is too complex and learns the noise or random fluctuations in the training data rather than the underlying patterns and relationships. Think of overfitting like trying to draw a curve through a set of points on a graph. If the curve is too complex, it fits the noise in the data rather than the underlying trend.

Overfitting is when a model is too complex and learns the noise or random fluctuations in the training data rather than the underlying patterns and relationships

Larger models with more parameters perform better because they can capture more subtle patterns and relationships in the data, leading to better performance on a wide range of tasks. On the other hand, larger the model, the more computational resources you are going to need—more memory, processing power, and training data to learn effectively, which can be a challenge for smaller organizations or those with limited resources.

Parameter size isn't the only factor you should consider, though. There are a few other things you should look for when picking a model.

You should consider the architecture of the model, including the type of layers, activation functions, and attention mechanisms.

You should consider the training data used to build the model: The old adage applies, garbage in, garbage out.

Certain models are just better at certain things. For example, some models are great at code analysis, whereas others are great at translations, etc.

So, let's say we have Code Llama. Code Llama is a model for generating and discussing code. It's built on top of Llama 2 and it's designed to make workflows faster and more efficient for developers, and make it easier for people to learn how to code. It can generate both code and natural language about code. And it supports many popular programming languages, such as Python, C++, Java, PHP, TypeScript (JavaScript), C#, Bash, and more.

Tell me, would you find it useful if you had a locally running model that you could point your code to and it explained in English what the code does? Would that be helpful for generating documentation or comments? Or perhaps reviewing pull requests? What about when you're working with a team that insists on writing in Java, but you're a C# person. And you need to convert a complex routine your buddy in the other team wrote from Java into C#. Code Llama can help.

Or what if you're dealing with something trivial yet frustrating, like writing a jQuery syntax to select every alternate row in a table. You know what? I downloaded Code Llama and ran this exact prompt, and here's what it gave me:

```
// Select every even row
$("table tr").filter(":even");

// Select every odd row
$("table tr").filter(":odd");
```

I don't know about you, but I'm impressed. How about a more complex task, this time to Llama3.1.

Write an Azure CLI script that provisions three Windows Server 2022 virtual machines behind a load balancer and that exposes port 443.

Honestly, I didn't have high hopes this would work. But it did. I wanted to paste the whole script here for you to see and review, but my editor insists I break every code listing into 70 characters or less, so I just said:

Please break down this script so it is of column width 70.

And it did. I've pasted the script in **Listing 1** and you be the judge! Frankly, my socks were blown off, I'm never writing another script by hand again, and unfortunately

Listing 1: Azure CLI script for a relatively complex task

```
#!/bin/bash

# Set variables
RESOURCE_GROUP="myResourceGroup"
LOCATION="eastus"
VM_NAME_PREFIX="myVm"
VM_COUNT=3
LOAD_BALANCER_NAME="myLoadBalancer"
VIRTUAL_NETWORK_NAME="myVirtualNetwork"
SUBNET_NAME="mySubnet"
PUBLIC_IP_ADDRESS_NAME="myPublicIpAddress"
SECURITY_GROUP_NAME="mySecurityGroup"

# Create resource group
az group create \
  --name $RESOURCE_GROUP \
  --location $LOCATION

# Create virtual network and subnet
az network vnet create \
  --resource-group $RESOURCE_GROUP \
  --name $VIRTUAL_NETWORK_NAME \
  --address-prefix 10.0.0.0/16 \
  --subnet-name $SUBNET_NAME \
  --subnet-prefix 10.0.1.0/24

# Create public IP address
az network public-ip create \
  --resource-group $RESOURCE_GROUP \
  --name $PUBLIC_IP_ADDRESS_NAME \
  --allocation-method Dynamic

# Create load balancer
az network lb create \
  --resource-group $RESOURCE_GROUP \
  --name $LOAD_BALANCER_NAME \
  --sku Basic \
  --frontend-ip-name $LOAD_BALANCER_NAME \
  --public-ip-address $PUBLIC_IP_ADDRESS_NAME

# Create backend pool
az network lb address-pool create \
  --resource-group $RESOURCE_GROUP \
  --lb-name $LOAD_BALANCER_NAME \
  --name myBackendPool

# Create health probe
az network lb probe create \
  --resource-group $RESOURCE_GROUP \
  --lb-name $LOAD_BALANCER_NAME \
  --name myHealthProbe \
  --protocol Tcp \
  --port 443

# Create load balancer rule
az network lb rule create \
  --resource-group $RESOURCE_GROUP \
  --lb-name $LOAD_BALANCER_NAME \
  --name myRule \
  --protocol Tcp \
  --frontend-port 443 \
  --backend-port 443 \
  --frontend-ip-name $LOAD_BALANCER_NAME \
  --backend-pool-name myBackendPool \
  --probe-name myHealthProbe

# Create security group
az network nsg create \
  --resource-group $RESOURCE_GROUP \
  --name $SECURITY_GROUP_NAME

# Create security rule for port 443
az network nsg rule create \
  --resource-group $RESOURCE_GROUP \
  --nsg-name $SECURITY_GROUP_NAME \
  --name myRule \
  --priority 100 \
  --source-address-prefix '*' \
  --destination-address-prefix '*' \
  --destination-port-range 443 \
  --access Allow \
  --protocol Tcp

# Create virtual machines
for i in $(seq 1 $VM_COUNT); do
  az vm create \
    --resource-group $RESOURCE_GROUP \
    --name ${VM_NAME_PREFIX}${i} \
    --image MicrosoftWindowsServer:
      WindowsServer:2022-Datacenter:latest \
    --size Standard_DS2_v2 \
    --vnet-name $VIRTUAL_NETWORK_NAME \
    --subnet $SUBNET_NAME \
    --nsg $SECURITY_GROUP_NAME \
    --admin-username azureuser \
    --admin-password P@ssw0rd1234!
done

az network nic create \
  --resource-group $RESOURCE_GROUP \
  --name ${VM_NAME_PREFIX}${i}Nic \
  --vnet-name $VIRTUAL_NETWORK_NAME \
  --subnet $SUBNET_NAME \
  --lb-name $LOAD_BALANCER_NAME \
  --lb-address-pool myBackendPool
```

this probably also means I'm going to fail all LeetCode interviews going forward. Why are those interviews relevant anyway in today's world?

Anyway, back to models.

Playing with Llama3

Remember, models are not just about text, and it's not just GPTs. You have many interesting fun models around computer vision, audio, and so much more. If you're interested in checking out some of the models available to you, you should check out this website: <https://huggingface.co/models>.

Let's leave that for another day. For now, let's focus on asking the latest issue of CODE Magazine some basic questions.

To get started, go ahead and install Ollama using the big "Download" button here: <https://ollama.com>. Once Ol-

lama is downloaded, go ahead and drag-and-drop it to your applications folder and launch it. Once you launch it, you should see a Llama icon in your menu bar. You can also visit <http://localhost:11434> and it should show you a message saying "Ollama is running."

What next? Ollama is just the engine. The real fun begins when you start downloading large language models and using them. There are many models you can try. I'll use Llama3. To run Llama3, launch terminal, and issue the following command:

```
ollama run llama3
```

Once you issue the command, you should see the prompt like that shown in **Figure 1**.

If this is the first time you've run Llama3, it first downloads the model. The model is a few gigabytes, so it might

```
sahilmalik@MakhiMax ➤ ollama run llama3
>>> Send a message (/? for help)
```

Figure 1: Ollama running Llama3

take a few minutes. If you don't have the model locally, it first downloads the model, as can be seen in Figure 2. Alternatively, you can set up the model ahead of time by issuing the following command:

```
ollama pull gemma2
```

Now that the model is running, let's ask it some questions. For instance, "What kind of oil does a 2023 Toyota Prius need?" You can see the results in Figure 3. Different models will handle this question differently. But I must say, Llama3 handled this pretty well.

Now for fun, switch off your internet connection and ask this question again. Yay! It still works. This means that you're running this model 100% locally. In fact, launch your activity monitor and view GPU history, and you'll notice that your model is really making good use of your local GPU, as can be seen in Figure 4.

For fun, I asked another question, "Write me a poem about CODE Magazine", and here are the first four sentences of a 16-line poem it wrote for me.

*CODE Magazine, a tale so fine,
A haven for coders, where knowledge is divine.
Within its pages, wisdom abounds,
As developers share their secrets, and coding bounds.*

Okay, this is getting to be real fun. "Write me a joke about Rod Paddock."

Why did Rod Paddock bring a ladder to the SQL Server conference?

Because he wanted to take his queries to the next level!

Okay, I can't let this go without taking a dig at Markus, so here goes.

```
sahilmalik@MakhiMax ➤ ollama run llama3
>>> What kind of oil does a 2023 toyota prius need?
The Toyota Prius, including the 2023 model year, typically requires a synthetic or synthetic-blend motor oil. Here are some specific recommendations:

* Toyota recommends using the following oils:
  + 0W-20 synthetic oil (Toyota Genuine Motor Oil)
  + 5W-30 synthetic blend oil (such as Mobil 1 or Castrol Edge)
* Other suitable options include:
  + API SN or SN Plus certified synthetic or synthetic-blend motor oil
  + ILSAC GF-5 or GF-6 certified synthetic or synthetic-blend motor oil

When choosing an oil, make sure to check the viscosity and certification specifications to ensure they meet Toyota's recommendations. You can also consult your vehicle's owner's manual or the manufacturer's website for more information.

Remember to always use the correct type of oil for your vehicle, as specified in the owner's manual, to maintain optimal performance and extend the life of your engine.

>>> Send a message (/? for help)
```

Figure 3: Llama3 answers a totally random question.

```
sahilmalik@MakhiMax ➤ ollama run gemma2
pulling manifest
pulling ff1d1fc78170... 100%
pulling 109037bec39c... 100%
pulling 097a36493f71... 100%
pulling 2490e7468436... 100%
pulling 10aa81da732e... 100%
verifying sha256 digest
writing manifest
removing any unused layers
success
>>> Send a message (/? for help)
```

Figure 2: Ollama pulling down a model for first time use

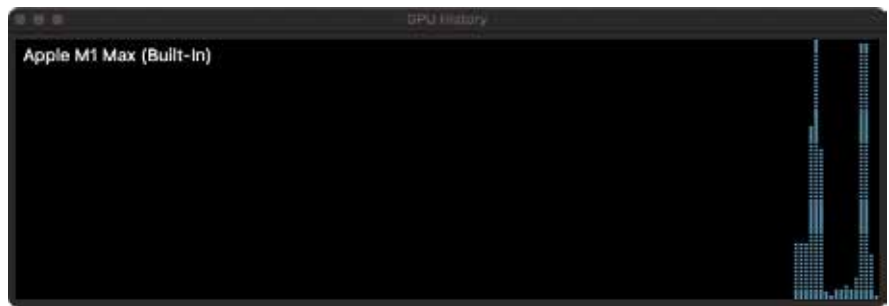


Figure 4: Llama3 making use of my GPU

Why does Markus have big muscles? Give me a funny answer

Markus has big muscles because he spent years lifting weights made of sauerkraut cans and doing squats on top of giant pretzels. Now, whenever he gets pumped up, his biceps start to glow with an otherworldly aura, making him the envy (and terror) of all the veggie platters in the land!

Ho ho ho! I should just say, "Write me a CODE Magazine article about AI." Just kidding. I wrote this article by hand—or did I? Hmm.

You'll note that Llama3 isn't up to date on the latest facts and figures. For instance, ask it "What is the current stock price of MSFT" and it will either show you old information, or will just point you to online resources. Well, that's not

great! What if I wanted to feed it real-time information, such as my company's HR policy around cracking jokes, so I could ask questions of it? You know, something useful, something a developer would need to do.

Putting On My Developer Hat

Okay, you can continue to have fun with Ollama. As long as you're happy with querying a world with only the information from when the model was created, it'll work great for you. Some facts and figures may never change, like how tall was Abraham Lincoln? Alas, he will never be any taller than he was. But new knowledge keeps getting created. And knowledge in my context is important.

What I want to do is access this model programmatically. I want to create a simple RAG application using LangChain, so I can feed in a PDF and ask questions relevant to it.

RAG

I hate acronyms, so let's define this new acronym. Retrieval-Augmented Generation (RAG) is the process of optimizing the output of a large language model, so it references an authoritative knowledge base outside of its training data sources before generating a response.

Large language models like Llama3 have been trained on vast volumes of data and use billions of parameters to generate original output for tasks. They can do many things, like answering questions, translating languages, and completing sentences. But they can't understand your domain and your data out of the box.

RAG extends the already powerful capabilities of LLMs to specific domains or an organization's internal knowledge base. Best of all, you can do so without having to pay for retraining the model. This makes it a very cost-effective way of extending an LLM so it becomes useful for your context.

To start, ensure that Ollama is running and create a new Python project. I won't delve into the specifics of Python here, so at a high level, I created a venv (virtual environment) with Python 3x, and I installed the following requirements:

```
langchain
langchain_community
pypdf
docarray
```

This is a matter of creating a requirements.txt with the above text and running the following command in terminal in your virtual environment:

```
pip install -r requirements.txt
```

With these requirements installed, I created an index.py file where I can start writing some code.

Let's start with something simple. Can I even call the local Llama3 model and ask it simple questions programmatically?

To begin, add the following imports to your code:

```
from langchain_community.llms
import Ollama
```

```
from langchain_community.embeddings
import OllamaEmbeddings
```

The first import you see is to a Python package called langchain_community. This package establishes a commonality between several third-party integrations via a common set of base interfaces. So, for instance, I'm importing llms.Ollama. I could just as easily import llms.OpenAI or llms.openai.AzureOpenAI, and many other such examples. This way, I can write code that's similar no matter what model I import. I could then write applications that go to OpenAI when online and to local Ollama when offline.

The second package being imported is langchain_community.embeddings. Embeddings in AI is a way of representing high-dimensional data as a set of vectors in a lower-dimensional space. These vectors, or embeddings capture the relationships between different data points in the original high-dimensional space. Think of it this way: When you're navigating through a city, a high-dimensional space is quite complex—it's a detailed 3D model of the city you're trying to find your way through. In this complex model, every street, every building, every restaurant, every place of interest is a point. This representation, although complete, can be very difficult to work with. Embeddings are a simplified map that captures all the essential relationships between these points. They make this data easier to work with by reducing the dimensionality of the data, while preserving the information you care for. This means that you get faster computation and better performance.

In terms of language models, you may want to think in terms of words as vectors in a way that captures their semantic meaning and relationships. Computer algorithms can capture this information by analyzing lots and lots of text in any given natural language and coming up with common occurrences of words and their relative interrelationships. So the word "deck" can have multiple meanings, but when you say PowerPoint deck or deck behind a house, the meaning of the word becomes clearer.

Using this model and embeddings in code is quite simple, as can be seen below:

```
MODEL="llama3"
model = Ollama(model=MODEL)
embeddings = OllamaEmbeddings(model=MODEL)
```

Once you've instantiated the object, using it is quite simple. I chose to invoke it with a simple question and gave it some additional hints to give me a short but funny answer (below). Really, I'm not interested in the biological aspects of eggs, even though it would be quite eggstraordinary. Just give me the funny bits and keep it short please. This can be done by invoking the model with an input text, as can be seen below.

```
out = model.invoke("
    What came first, chicken or egg?
    Give me a funny and short answer.")
print(out)
```

Running this gives me a simple output as below.

The age-old question!

Well, let's get to the bottom of this fowl play (get it?). According to expert sources (okay, I made them up), it was actually the egg that came first.

Why, you ask? Well, because dinosaurs used to lay eggs, and then they evolved into chickens. So, in a nutshell (or an eggshell?), the egg came before the chicken!

Well, that was egg-cellent. You can find the full code for this initial very simple example in **Listing 2**.

I am good at puns with the word "eggs," aren't I? Well, here is an egg-cellent way of finding great puns using the word "egg." Just use the language model to answer "Tell me some pun words using the word egg." Honestly, this saved me a lot of mental egg-cercise.

Okay fun stuff! Let's take this a bit further now.

As you can see, I'm doing "model.invoke." But the word LangChain has the word "chain" in it. The idea is that I can build a chain for processing my output. For instance, I could have the output returned in any format. But I want to always see the output as a string, so I can print it out.

To achieve this, I'm going to import a parser, as shown below.

```
from langchain_core.output_parsers
import StrOutputParser
```

Just like rest of the LangChain ecosystem, these parsers have been written in to define a base class so they can be used across many LLMs. With the parser imported, now I can build my chain as follows, and slightly modify my code:

```
parser = StrOutputParser()
chain = model | parser
out = chain.invoke(
    "Give me some puns using the word egg")
print(out)
```

By doing so, I'm effectively still doing model.invoke, passing the output of that to the parser, and then writing out the results. Go ahead and run the application, you should see some egg-strordinary results. You can find the code for involving a chain in the LLM code in **Listing 3**.

Now that you have a good foundation to build upon, you can take the application further. You can now create increasingly complex chains, so the next thing you need to do is add prompts. In fact, what you want to do next is, with the help of prompts, get the model to not use its existing knowledge, but instead provide it some context, and the answer must be given based on that input context. If the model has no idea how to answer a given question based on the context given, the model should simply say "I don't know!" The code for this can be seen in **Listing 4**. Let's walk through it.

The first thing you see in **Listing 4** is a prompt template. A prompt template is simply a set of parameters that the user can specify, which can then be used to generate a

Listing 2: Calling an LLM using code

```
from langchain_community.llms import Ollama
from langchain_community.embeddings
import OllamaEmbeddings

MODEL="llama3"
model = Ollama(model=MODEL)
embeddings = OllamaEmbeddings(model=MODEL)

out = model.invoke(
    "What came first, chicken or egg?
    Give me a funny and short answer.")
print(out)
```

Listing 3: Invoking the LLM using a chain

```
from langchain_community.llms import Ollama
from langchain_community.embeddings
import OllamaEmbeddings
from langchain_core.output_parsers
import StrOutputParser

MODEL="llama3"
model = Ollama(model=MODEL)
embeddings = OllamaEmbeddings(model=MODEL)

parser = StrOutputParser()
chain = model | parser
out = chain.invoke(
    "Give me some puns using the word egg")
print(out)
```

Listing 4: Answering questions based on an input context

```
template = """
Answer the question based on the context below.
If you can't answer the question, say "I don't know".

Context: {context}
Question: {question}
"""

prompt = PromptTemplate.from_template(template)
prompt.format(
    context="Here is some context",
    question="Here is a question")
parser = StrOutputParser()
chain = prompt | model | parser
out = chain.invoke(
    {"context":
    "Glyphosates can often cause deadly
    cancers which can kill people.",
    "question": "Who is Abraham Linclon?"})
print(out)
```

prompt for the language model. The idea is that you want to give some guidelines to the model, so you can hopefully get a more intelligent answer. Give it a pre-defined structure or format for inputting text.

Let's examine the template a bit closer

```
Answer the question based on the context below.
If you can't answer the question,
say "I don't know".
```

```
Context: {context}
Question: {question}
```

With a well-designed prompt template, you can clarify what you wish to achieve. Okay, so you're instructing the model to provide an answer with the given context, but

not attempt to make up stuff it doesn't know with a degree of confidence.

With the prompt template, you can also provide context. That's the {context} placeholder you see. This is a way of providing relevant information that helps the model provide a better answer. This can be historical information, previous chats, or, as you'll see shortly, input from a CODE Magazine article, based on which I'd like to have questions answered.

With a prompt template, you can also tune the tone and style. For example, in the above template, I can simply add the text "Assume the tone of a comedian," and the replies will be funny.

With a prompt template, you can also provide guiderails to the answer. For example, if I don't want wordy answers, and I want key details listed out, I can add the following instructions to my prompt template:

```
Instructions:
Keep your answers to less than a 100 words
Summarize answers as bullet points
```

Then you chain the prompt into the LangChain as usual, and finally when you invoke the chain, you provide some input and ask a reasonable question.

Now, what's odd here is that the answer to "Who is Abraham Lincoln?" is fairly common knowledge. In fact, if I asked this question of Llama3 directly without context, it would confidently answer that "Abraham Lincoln (1809-1865) was the 16th President of the United States."

But, given the input context as shown in **Listing 4**, when I ask the same question, the model simply replies:

```
I don't know
```

Now let's ask some questions that the model can answer.

What are Glyphosates?

Glyphosates are a type of herbicide that can be toxic to humans and have been linked to an increased risk of cancer.

How do people die?

Based on the context, people die from the deadly cancers caused by glyphosate exposure.

What causes cancer?

Glyphosate

I have to say that I'm incredibly impressed. Notably, ask the same questions in the Llama3 model using Ollama and run Llama3, you'll get a much different and more detailed answer. That is because Ollama running plain vanilla Llama3 lacks context or the prompt we provided.

Now change the context to something clearly inaccurate. For instance, I changed my context to, "Glyphosates are delicious and can be added to make a cake sweet," and then I asked a question, "What makes cakes sweet?"

Before I tell you what the model would answer, I want you to look at the first question I asked of the model, where I simply asked "What are Glyphosates?" Note that my context never mentioned that Glyphosates are an herbicide. The model knew it was an herbicide. This is based on the massive amount of training the model has had, so it isn't giving you clearly inaccurate answers. Of course, given enough inaccurate context, and the model will start spewing out nonsense. Similarly, for the second question "How do people die?" the answer clearly says, "Based on the context..."

Take a step back and think for a second. What if I gave you, a human being, lots of inaccurate context. Would you say the earth is flat? At one point, a large percentage of population did believe the earth was flat. Heck. They killed Galileo over this argument, and the poor chap was right. It really makes you wonder what we're being lied to about today.

Anyway, so what makes cakes sweet? Per the context, it's Glyphosates, but the model based on its previous learning knows there's something off. And it replies as follows.

I think there may be some confusion here!

Glyphosate is actually an herbicide, commonly known as Roundup, used in agriculture. It's not typically used in baking or making cakes.

To answer your question, what usually makes cakes sweet is the addition of sugar, honey, maple syrup, or other sweeteners. The type and amount of sweetener can vary depending on personal preference and recipe.

So, to summarize: Glyphosate doesn't make cakes sweet!

Okay now that you can provide context and ask questions based on context, and you understand how this works, let's enhance it further. I'm next going to provide the last issue of CODE Magazine as input to my model. And then I can ask questions based on the last issue of CODE Magazine. The input format will be PDF. The code for this can be seen in **Listing 5**.

There are some things in **Listing 5** that are immediately familiar. The output parser being used, the template with the context, and the question are all concepts I've already discussed. The new part here is that I'm using PyPDFLoader to load up an input PDF, and creating a vector store using an in-memory search object.

PyPDFLoader is one of the many document loaders available in langchain_community. Each of these document loaders takes the task of converting a given input into documents that can be used to create a vector store. I encourage you to explore the various other document loaders available in langchain_community.document_loaders. For instance, a very common task you'll run into is to create a vector store directly from a website. Here is how you do that:

```
from langchain_community.document_loaders
import WebPageLoader
```

Listing 5: Chain a PDF into my langchain

```
loader = PyPDFLoader("CodeMagJulAug2024.pdf")
pages = loader.load_and_split()
vectorstore =
    DocArrayInMemorySearch.from_documents(pages,
        embedding=embeddings)
retriever = vectorstore.as_retriever()

docs = retriever.invoke("programming")
print(docs)

template = """
Answer the question based on the context below.
If you can't answer the question, say "I don't know".

Context: {context}
Question: {question}
"""

prompt = PromptTemplate.from_template(template)
parser = StrOutputParser()

chain = (
    {
        "context": itemgetter("question") | retriever,
        "question": itemgetter("question"),
    }
    | prompt
    | model
    | parser
)

exit = False
while not exit:
    question = input("Ask a question: ")
    if question == "bye":
        exit = True
    else:
        print(
            f"Answer: {chain.invoke({'question': question})}")
```

```
loader = WebPageLoader()
doc = loader.load("https://www.codemag.com")
```

Here's another fun loader for you to try. Check out the `langchain_community.document_loaders.blob_loaders.YoutubeAudioLoader`. Wouldn't it be fun to load up YouTube audio of a video and just ask a question? How many times have you seen an influencer talk about random clickbait stuff for the first nine minutes before getting to the point in the last minute of a 10-minute video? What about `WikipediaLoader`? Yes, that's there too. Try it out.

The next interesting thing you see in **Listing 5** is the line below.:

```
vectorstore =
    DocArrayInMemorySearch.from_documents(
        pages, embedding=embeddings)
```

For a simple PDF this is fine, but for larger sets of data you'll want to use something persistent. This means that every time you run the program, it starts at zero. And it may take a few minutes to ingest a PDF document, so this can get really annoying. For the sample application this is fine, but in the real world, you'll probably want to save the vector store, maybe remove documents from it, or add to it, without having to recalculate everything.

There are many ways to achieve persistence across runs. There are many available classes in `langchain_community.vectorstores` that help you target alternate storage locations. For example, you can use Chroma DB. To use Chroma DB, install the db in your project as follows:

```
pip install chromadb
```

Once Chroma DB is installed, you can import it into your Python code as follows:

```
from langchain_community.vectorstores import Chroma
```

Once imported, you can use Chroma DB to create a persistent store, as shown below:

```
vectorstore = Chroma.from_documents(
    pages, embedding=embeddings,
```

```
persist_directory="./codemag")
retriever = vectorstore.as_retriever()
```

The first time you run this code, it will take some time to crunch up the PDF. Once it's done with it, it will save all its work in a directory called "codemag". Next time you run the program, you can simply check for the existence of the codemag folder, and if it exists, simply load up the vector store, as shown below:

```
vectorstore = Chroma(
    persist_directory="./codemag",
    embedding_function=embeddings)
```

Because you skipped all the hard calculations, this loads almost instantly and is ready to use. The full code using Chroma DB and persistent storage can be seen in **Listing 6**. Anecdotally, when I ran this on my M1 Max, I had MS Word, VSCode, Chrome, and Safari running. The first time I ran this, it took me around seven minutes, and this was one of the rare occasions I heard my M1 Max's fan come on. Well, it's nice to know the fan works. If you own a MacBook Pro, you know the fan nearly never comes on. The second time, the load was nearly instantaneous and the results were the same.

From seven minutes to an instant start for a simple PDF? I'd call that an improvement. What does it mean when ingesting terabytes of data? You know, like that stupid HR manual that won't let me crack jokes at work.

Once you've created the vector store with the given embeddings, you create a retriever from it. Once you have a retriever, you can choose to invoke it. When you invoke it, you can give it some input context and it returns the top four documents from the input source for the specified input query to the `retriever.invoke` function. This is, of course, configurable.

Now I know I wrote some stuff about VSCode in the CODE Magazine article for July/August 2024, so when I invoke `retriever.invoke` with "VSCode" as an input parameter, indeed the retriever shows me four locations in the PDF where VSCode was most relevant. This can be seen in **Figure 5**. Of course, I knew this already because I diligently read the magazine cover to cover. Or did I? Hmm...

But wait! There's more! I can now chain the retriever in my chain, and now run a loop to allow the user to ask questions of my document in a free-form manner. As you can see from **Listing 5** until the user says "bye," you can, in a loop, allow the user to ask whatever questions the user wishes to ask.

Let's try a few inputs.

Who is a huge fan of VSCode?

Based on the context, it seems that the author of this text is a huge fan of VSCode. They mention using VSCode for various purposes, including taking meeting notes in markdown, and appreciate its features such as Emmet expansion, multiple cursors, and code selection.

How do you hide files in VS Code?

According to the context, you can hide files via a setting. For instance, the setting below in your `.vscode/settings.json` folder hides the `node_modules` folder from your view, even though it may exist on the disk.

```
```json
{
 "files.exclude": {
 "node_modules": true
 }
}
```

```
}
```
```

This setting can be found in your `.vscode/settings.json` file.

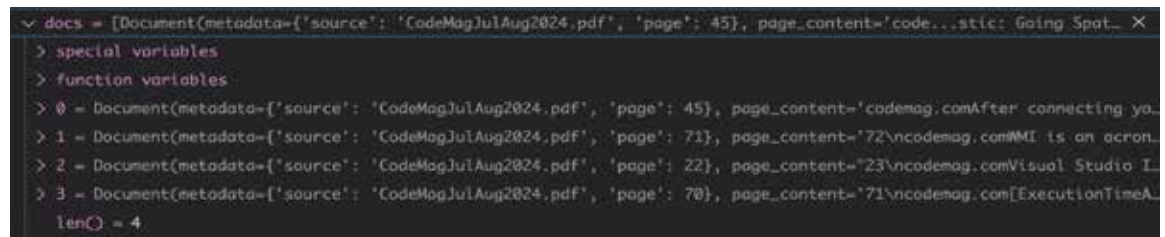
Try comparing the results you see in these questions with what's mentioned in the article. The answers are being pulled out of the input document you provided. Now, this knowledge about hiding files is public information. Try asking this question directly of Llama3, and you'll get a much more generic answer.

Now let's try asking for something that's only mentioned in the July/August article.

How do I create the CustomerController class?

Based on the provided context, it seems that you are implementing an ASP.NET Core application with MVC. To create the `CustomerController` class, you would typically follow these steps:
.. (detailed answer omitted for brevity)

Honestly, look at the article in its original form and compare this answer. It's almost like this answer gives some much more actionable information.



```
> docs = [Document(metadata={'source': 'CodeMagJulAug2024.pdf', 'page': 45}, page_content='code...stic: Going Spat... X
> special variables
> function variables
> 0 = Document(metadata={'source': 'CodeMagJulAug2024.pdf', 'page': 45}, page_content='codemag.comAfter connecting ya...
> 1 = Document(metadata={'source': 'CodeMagJulAug2024.pdf', 'page': 71}, page_content='72\ncodemag.comMMI is an acron...
> 2 = Document(metadata={'source': 'CodeMagJulAug2024.pdf', 'page': 22}, page_content='23\ncodemag.comVisual Studio I...
> 3 = Document(metadata={'source': 'CodeMagJulAug2024.pdf', 'page': 70}, page_content='71\ncodemag.com[ExecutionTimeA...
len(docs) = 4
```

Figure 5: VS Code returns results

Listing 6: Persisting a vectorstore across runs

```
import os
from langchain_community.llms import Ollama
from langchain_community.embeddings import OllamaEmbeddings
from langchain_core.output_parsers import StrOutputParser
from langchain.prompts import PromptTemplate
from langchain_community.document_loaders import PyPDFLoader
from langchain_community.vectorstores import Chroma
from operator import itemgetter

MODEL="llama3"
model = Ollama(model=MODEL)
embeddings = OllamaEmbeddings(model=MODEL)

folder_path = "./codemag"

if os.path.isdir(folder_path):
    vectorstore = Chroma(persist_directory=folder_path,
                        embedding_function=embeddings)
else:
    loader = PyPDFLoader("CodeMagJulAug2024.pdf")
    pages = loader.load_and_split()
    vectorstore = Chroma.from_documents(
        pages, embedding=embeddings,
        persist_directory=folder_path)

retriever = vectorstore.as_retriever()

docs = retriever.invoke("VSCode")

print(docs)

template = """
Answer the question based on the context below.
If you can't answer the question, say "I don't know".

Context: {context}
Question: {question}
"""

prompt = PromptTemplate.from_template(template)
parser = StrOutputParser()

chain = (
    {
        "context": itemgetter("question") | retriever,
        "question": itemgetter("question"),
    }
    | prompt
    | model
    | parser
)

exit = False
while not exit:
    question = input("Ask a question: ")
    if question == "bye":
        exit = True
    else:
        print(
            f"Answer: {chain.invoke({'question': question})}"
        )
```

Now is this model perfect? No. The input PDF had a lot of structure. For instance, nowhere does it say “Author of this article is .. xyz”, instead it just shows the author bio as a sidebar. This is where you could spend some effort tweaking input, including things like image recognition to make your data corpus more useful. You’ll find methods on the vectorstore object to add meaningful contexts to make the LLM more useful. Think of it like the “best bets” feature in a search engine. If you see your users asking the same question over and over again and the LLM giving a less than ideal answer, just teach your LLM a bit by adding that context in plain English. Maybe in my next article I’ll do it in Spanish too, all using AI of course.

But if you’re faced with a large tome of boring unstructured text and you want to quickly make sense of it, this is a fantastic way of achieving that goal.

Wouldn’t it be nice, if you were reading the script of a complex TV series like, say, Game of Thrones, and getting really confused between all those characters and mythical lands, and you could simply ask a question like:

Who were the Lannisters?

According to the context, the Lannisters are a noble family from Casterly Rock, and the following individuals are mentioned as being part of this family:

1. Tyrion (also known as Imp or Halfman), who is the narrator and a dwarf.
2. Tywin (Tyrion’s father), who was the Hand of the King for twenty years.
3. Cersei (Tyrion’s sister), who is married to King Robert Baratheon.
4. Kevan Lannister (Cersei’s brother), who is mentioned as part of the party traveling with King Robert.

You can imagine what an unimaginable leap this is for me. I always fall asleep halfway into each episode, I can finally wrap my head around all those characters and appear somewhat knowledgeable. So when I’m suddenly woken up because the Mrs. called the water works over some emotional scene, I can consult my LLM to quickly get up to speed with what’s going on, and precisely calculate the consoling time required before we can switch to the sports channel. AI is good for my social life.

As impressive as this is, remember, all this is running locally on my very simple off-the-shelf Mac. You can easily tweak this to use OpenAI and GPT4, and the answers will be far more accurate. But it’s only a matter of time before local models become more powerful and, with more fine tuning and better prompt engineering, locally running models are very useful at this point. This is why I like Apple’s approach to AI so much: because they have split AI responsibilities to local first and provisioned in the cloud on-demand when needed.

Summary

I’ve been in this industry for a few decades now, and when I saw the first gigabyte hard disk roll around, I was floored at the amount of storage it had in such a small space. I started doing some research on how much stor-

age a human mind has. How much compute does a human mind have? I knew right there that, within my lifetime, I’ll see computers have more storage and compute power than a human being. The cloud was unimaginable back then, but the cloud has brought all that reality to the forefront much sooner than I’d imagined.

What’s the storage and compute power of the cloud? And what will we do with it?

What I find even more amazing though, is how much power our local devices have. The next version of iOS will do a bunch of AI compute locally and punt to the cloud where necessary. The newest ARM chips that Windows PCs run on have an NPU, a neural processing unit. The pixel phone has always been more about the software than the hardware.

Sure, Windows ships with some features that leverage AI, but I truly feel the real power will be unlocked by developers around the world once these PCs with NPUs are commonplace and there’s a serious developer story around them. Even today, we’re only beginning to tap into the value of local AI.

Let’s fast forward a bit. In a world where every local device will have local AI capabilities, what will using a computer be like? Will I be able to ask my computer to sift through all the pictures of a loved one I lost, create a vision depth video, and let me relive the moments with an Apple Vision Pro and literally interact with an absent loved one in a manner so convincing that reality and imagination start to blend together?

Creepy? Or cute? You decide.

I can tell you, when photographs were invented, people had the same misgivings. People didn’t want their picture taken because they were afraid the camera captures their soul. And yet here we are.

Pair this with AR (augmented reality). What I wouldn’t give to call out to my dog, who unexpectedly passed away at the young age of four, just one more time, and have him come running to me, like he always did, with a ball in his mouth.

Would it not be truly amazing to have AR glasses that let me zoom, autocorrect for my vision, see in the dark, and that shows me notifications and directions, uses AI and measured biometrics to ask me to chill a bit and saves me from a heart attack, detects important biomarker changes, and persuades me to get screened for cancer?

Or how populations’ minds will be controlled by feeding them convincing but incorrect stories, literally making “seeing is believing” obsolete. What are the implications of such power used for good or for bad?

The possibilities are both scary and feel like a next-level evolutionary leap. I’ll see it, all in this lifetime.

What an incredible time to be around. Until the next time!

Sahil Malik
CODE

SPONSORED SIDEBAR

Adding Copilots to Your Apps

The future is here now and you don’t want to get left behind. Unlock the **true potential** of your software applications by adding **Copilots**.

CODE Consulting can assess your applications and provide you with a roadmap for adding Copilot features and optionally assist you in adding them to your applications.

Reach out to us today to get your application assessment scheduled.
www.codemag.com/ai

Exploring .NET MAUI: Data Entry Controls and Data Binding

In the first parts of this ongoing series on exploring .NET MAUI (<https://codemag.com/Article/2408041/Exploring-.NET-MAUI-Getting-Started> and <https://codemag.com/Article/2409041/Exploring-.NET-MAUI-Styles-Navigation-and-Reusable-UI>), you created your first .NET MAUI application and ran that application on both a Windows computer and an Android emulator.



Paul D. Sheriff

<http://www.pdsa.com>

Paul has been working in the IT industry since 1985. In that time, he has successfully assisted hundreds of companies' architect software applications to solve their toughest business problems. Paul has been a teacher and mentor through various mediums such as video courses, blogs, articles and speaking engagements at user groups and conferences around the world. Paul has multiple courses in the www.pluralsight.com library (<https://bit.ly/3gvXgvj>) and on YouTube.com (<https://www.youtube.com/@pauldsheff>) on topics ranging from C#, LINQ, JavaScript, Angular, MVC, WPF, XML, jQuery, and Bootstrap. Contact Paul at psheff@pdsa.com.



You created data-entry pages, partial pages to reuse on those pages, and you navigated among those pages. In this article, you'll continue to use more data entry controls and learn to perform data binding between controls. You're going to create a class with properties that you can bind to controls on a page as well. When you change the values of properties in a class, you need to raise a `PropertyChanged` event so the UI can update those controls that are bound to the properties. You're going to create a base class that helps you raise that event any time the property values change.

Use a Switch Control for Yes/No Input

Simple entry controls allow a user to enter any data they want. For some input you need a simple yes or no answer from the user. For example, if you have a Boolean property such as `IsEmployed` on a business object, use a Switch control to represent the two states for this property. On a Windows computer, the Switch control appears as a toggle coupled with a label next to it with the words **On** and **Off**, as shown in **Figure 1**.

Open the `Views\UserDetailView.xaml` file and add an **Auto** to the `RowDefinitions` attribute of the Grid control. Locate the last `<HorizontalStackLayout>` starting tag at

the bottom of the Grid and change the `Grid.Row` property to "7" instead of "6". Just before the `<HorizontalStackLayout>` starting tag you just modified, add the following XAML that creates the Switch control:

```
<Label Grid.Row="6"
      Text="Still Employed?" />
<Switch Grid.Row="6"
        Grid.Column="1" />
```

Try It Out

Run the application and click on **Users > Navigate to Detail** to see the switch control. Click on the Switch control a couple of times to see the labels change.

Override the Switch Labels

The labels **On** and **Off** may not make sense for what you're trying to represent with the Switch control. In the example above, it makes more sense for the labels to be **Yes** and **No**. You may also not want the labels to appear at all when running on Windows. There's no exposed property to set these two labels, but you may write a little C# code to set these labels.

In the first part of this article series (<https://codemag.com/Article/2408041/Exploring-.NET-MAUI-Getting-Started>), you added a `SetWindowsOptions()` method to the `MauiProgram.cs` file. Around this method, you added a conditional compile statement `#if WINDOWS`. Instead of using these conditional compile statements around each method that you want to run on only a specific platform, let's create a class in the `Platforms\Windows` folder instead. When a class is created within any of the sub-folders under the `Platforms` folder, that class is only compiled for running on that specific OS, thus you don't need to use conditional compilation statements.

Right mouse-click on the `Platforms\Windows` folder and add a new class named `WindowsHelpers`. Into this class, you're going to place the `SetWindowOptions()` method you created in the first article, and a new method named `SetSwitchText()`. This new method allows you to set the `OnContent` and `OffContent` properties after a Switch control has been rendered on a .NET MAUI page. Replace the entire contents of the `WindowsHelpers.cs` file with the code shown in **Listing 1**.

Open the `MauiProgram.cs` file and in the `CreateMauiApp()` method where you called the `SetWindowOptions()` method, modify it to use `WindowsHelpers.SetWindowOptions(builder)` instead. Also, call the `SetSwitchText()` method on the same class as shown below:

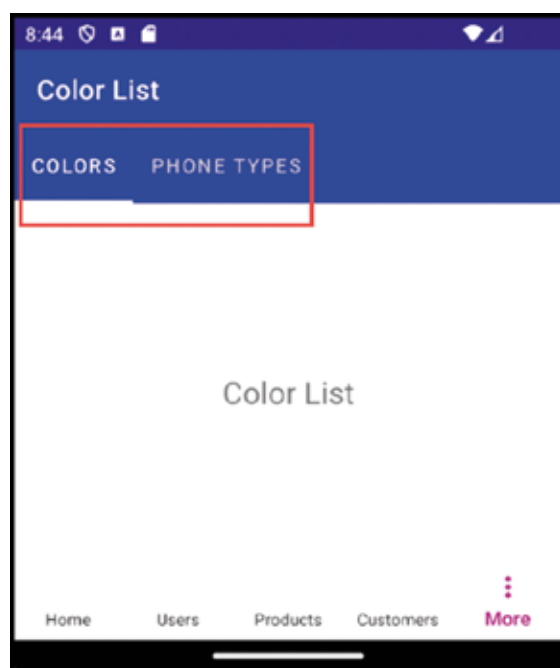


Figure 1: A switch represents a Boolean state.

Listing 1: Create a WindowsHelpers class to change Windows-only settings

```
using Microsoft.Maui.Handlers;
using Microsoft.Maui.LifecycleEvents;
using Microsoft.UI;
using Microsoft.UI.Windowing;
using WinRT.Interop;

namespace AdventureWorks.MAUI;

public class WindowsHelpers {
    #region SetWindowOptions Method
    public static void SetWindowOptions(
        MauiAppBuilder builder) {
        builder.ConfigureLifecycleEvents(events => {
            events.AddWindows(wndBuilder => {
                wndBuilder.OnWindowCreated(window => {
                    IntPtr hwnd = WindowNative
                        .GetWindowHandle(window);
                    WindowId winId = Win32Interop
                        .GetWindowIdFromWindow(hwnd);
                    AppWindow appWnd = AppWindow
                        .GetFromWindowId(winId);
                    if (appWnd.Presenter is
                        OverlappedPresenter p) {
                        p.Maximize();

                        //p.HasBorder = false;
                        //p.HasTitleBar = false;

                        //p.IsAlwaysOnTop = false;
                        //p.IsMaximizable = false;
                        //p.IsMinimizable = false;
                        //p.IsModal = false;
                        //p.IsResizable = false;
                    }
                });
            });
        });
    }
    #endregion

    #region SetSwitchText Method
    public static void SetSwitchText(
        string onContent = "",
        string offContent = "") {
        SwitchHandler.Mapper
            .AppendToMapping("SwitchText", (h, v) => {
                // Get rid of On/Off label beside switch
                h.PlatformView.OnContent = onContent;
                h.PlatformView.OffContent = offContent;

                h.PlatformView.MinWidth = 0;
            });
    }
    #endregion
}
```

```
#if WINDOWS
WindowsHelpers.SetWindowOptions(builder);

// Set labels on all <Switch> elements
WindowsHelpers.SetSwitchText("Yes", "No");
//WindowsHelpers.SetSwitchText();
#endif
```

Remove the `SetWindowOptions()` method in the `MauiProgram.cs` file and the `using` statements wrapped in the `#WINDOWS` conditional compile statement at the top of the file.

Try It Out

Run the application and click on **Users > Navigate to Detail** to see the Switch control with the labels **Yes** and **No**, as shown in **Figure 2**. If you don't pass any parameters to the `SetSwitchText()`, no labels are displayed next to the Switch at all.

RadioButton Controls Help Users Select a Single Item from a List

If you have a small set of items that the user can select a single value from, the `RadioButton` control is one of a few controls you can use. The `RadioButton` controls (**Figure 3**) are a mutually exclusive set of inputs. When the user clicks on one `RadioButton`, the previously selected button is unchecked and the one newly clicked becomes checked. You may set the initial button to be checked by setting the **IsChecked** property to a true value. Each set of `RadioButton` controls should have the same **GroupName** property set to a unique value. For example, **GroupName="EmployeeType"** is used to group those `RadioButton` controls shown in **Figure 3**. If you had a set of `RadioButton` controls to group a user's ethnicity, you set the **GroupName** property of those controls to **GroupName="Ethnicity"**.

Open the `Views\UserDetailView.xaml` file and add an **"Auto"** to the **RowDefinitions** attribute of the `Grid` control. Locate the last `<HorizontalStackLayout>` starting tag

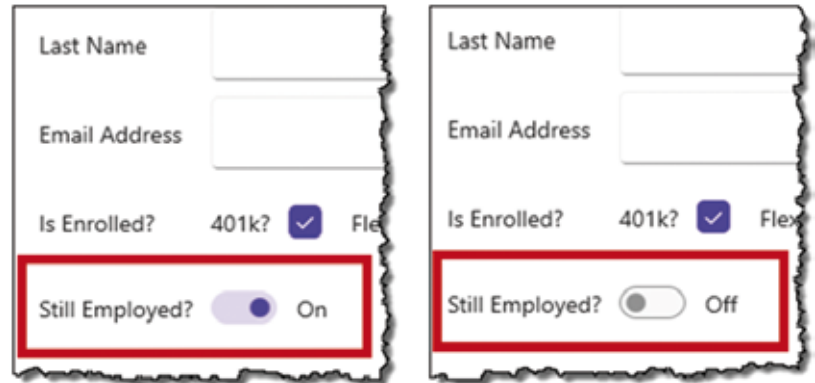


Figure 2: You can set the labels on the Switch to be anything you wish.

at the bottom of the `Grid` and change the **Grid.Row** property to "8" instead of "7". Just before the `<HorizontalStackLayout>` starting tag you just modified, add the following XAML that creates the set of `RadioButton` controls:

```
<Label Text="Employee Type"
    Grid.Row="7" />
<FlexLayout Grid.Row="7" Grid.Column="1"
    Wrap="Wrap" Direction="Row">
    <HorizontalStackLayout>
        <Label Text="Full-Time" />
        <RadioButton IsChecked="True"
            GroupName="EmployeeType" />
    </HorizontalStackLayout>
    <HorizontalStackLayout>
        <Label Text="Part-Time" />
        <RadioButton GroupName="EmployeeType" />
    </HorizontalStackLayout>
</FlexLayout>
```

Try It Out

Run the application and click on **Users > Navigate to Detail** to see the radio buttons, as shown in **Figure 3**. Click



Figure 3: Radio buttons allow you to select a single value from a list of items.



Figure 4: A full calendar is displayed from which the user may select a date.

back and forth between the two `RadioButton` controls to see the other one become unchecked.

The DatePicker Control Avoids Typing in Date Values

It's always better to let users select from a list rather than typing in something by themselves. The `DatePicker` control displays a calendar from which the user may select a date. The `Date` property gets the date selected by the user or sets the date to start with when the calendar is first displayed. By default, the `Date` property is set to today's date. The `Format` property is a string value set to a standard or custom .NET format string you pass to the `ToString()` method, such as `dateValue.ToString("D")`. The default format string is set to the long date pattern "D". There are two properties, `MinimumDate` and `MaximumDate`, that allow you to set the range of dates the user is allowed to select from.

Let's add a Birth Date field to the user detail page and use the `DatePicker` control to allow the user to select the date from the calendar. Open the `Views\UserDetailView.xaml` file and add an "Auto" to the `RowDefinitions` attribute of the `Grid` control. Locate the last `<HorizontalStackLayout>` starting tag at the bottom of the `Grid` and change the `Grid.Row` property to "9" instead of "8". Just before the `<HorizontalStackLayout>` starting tag you just modified, add the following XAML that creates the `DatePicker` control.

```
<Label Text="Birth Date"
      Grid.Row="8" />
<DatePicker Grid.Row="8"
            Grid.Column="1" />
```

Try It Out

Run the application and click on **Users > Navigate to Detail** to click on the Birth Date entry field. A calendar control is displayed, as shown in Figure 4.

Select an Hour and Minute Using the TimePicker Control

The `TimePicker` control displays a list to the user from which they may select an hour and minute (see Figure 5). The `Time` property gets the time selected by the user or sets the time to start with when the list of hours and minutes is first displayed. The `Time` property is a `TimeSpan` data type. By default, the `Time` property is set to zero (0) or midnight (12:00:00 AM). The `Format` property is a string value set to a standard or custom .NET format string you pass to the `ToString()` method, such as `timeValue.ToString("t")`. The default format string is set to the short time pattern "t". Be aware that on a macOS, the `Format` property has no effect on the control.

Let's add a Start Time field to the user detail page and use the `TimePicker` control to allow the user to select the time from an hour and minute list, as shown in Figure 5. Open the `Views\UserDetailView.xaml` file and add an "Auto" to the `RowDefinitions` attribute of the `Grid` control. Locate the last `<HorizontalStackLayout>` starting tag at the bottom of the `Grid` and change the `Grid.Row` property to "10" instead of "9". Just before the `<HorizontalStackLayout>` starting tag you just modified, add the following XAML that creates the `TimePicker` control.

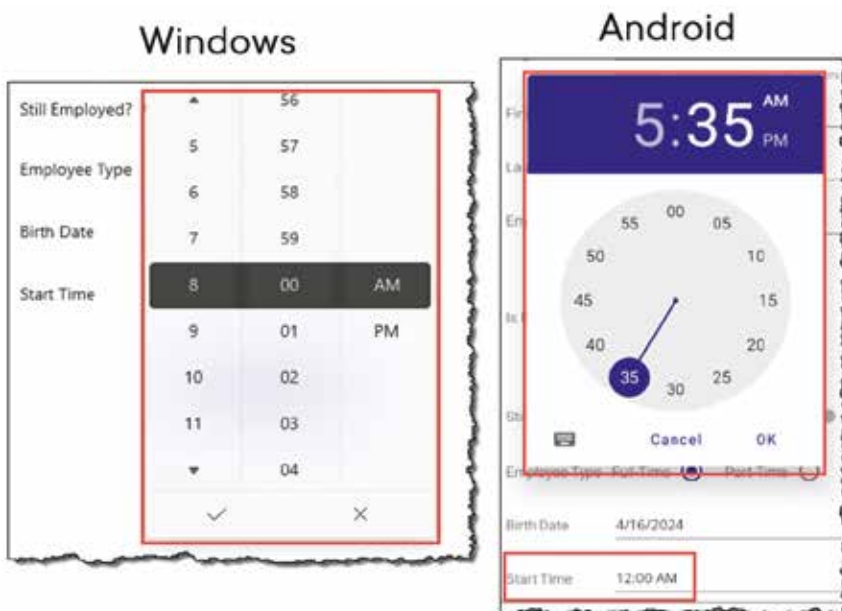


Figure 5: Use a `TimePicker` to select the hour and minute from an hour and minute list.



Figure 6: Use a Picker control to display a list of items from which to select.

```
<Label Text="Start Time"
      Grid.Row="9" />
<TimePicker Grid.Row="9"
            Grid.Column="1"
            Time="08:00:00" />
```

Try It Out

Run the application and click on **Users > Navigate to Detail**. Then click on the start time entry field to have an hour and minute list from which you can select, as shown in **Figure 5**.

Choose an Item from a List with the Picker Control

If you have a small set of values the user can select from but not a lot of real estate on your page for radio button controls, a Picker control can be an appropriate choice. The Picker control has an **ItemsSource** property that can be set to any **IEnumerable** collection. The **SelectedIndex** property is an integer value representing the index number within the collection that's selected. The **SelectedItem** property represents the actual item selected. Most often the **ItemsSource** property is set at runtime via data binding, but for this sample, you're going to use a hard-coded array of strings.

Let's add a Label and Entry field to enter a Phone number. Next to the Phone entry field, display a list of phone types the user can select from, as shown in **Figure 6**. Open the **Views\UserDetailView.xaml** file and add an **"Auto"** to the **RowDefinitions** attribute of the Grid control. Locate the last **<HorizontalStackLayout>** starting tag at the bottom of the Grid and change the **Grid.Row** property to "11" instead of "10". Just before the **<HorizontalStackLayout>** starting tag you just modified, add the XAML shown in **Listing 2** that creates the Label, Entry, and the Picker control.

Within the **<Picker>** element, create a **<Picker.ItemsSource>** element into which you place a collection of phone types. The collection is created using the **x:Array** construct where each type in the array is of the type **String**. For a small set of data that doesn't change very often, this is acceptable, but getting this data from a data store is probably a better option. Data binding is covered later in this article series, and you will learn to set the **ItemsSource** property using data from a data store.

Try It Out

Run the application and click on **Users > Navigate to Detail** and click on the down arrow on the right of the

Picker to see the list of phone types appear. Select one of the options to watch the list close and the selected item appear next to the phone number entry.

Listing 2: Enter a phone number and select the type of phone from a drop-down list

```
<Label Text="Phone"
      Grid.Row="10" />
<FlexLayout Grid.Row="10"
            Grid.Column="1"
            Wrap="Wrap"
            Direction="Row">
  <HorizontalStackLayout>
    <Entry MinimumWidthRequest="150"
          Text="" />
  </HorizontalStackLayout>
  <HorizontalStackLayout>
    <Picker VerticalTextAlignment="Center">
      <Picker.ItemsSource>
        <x:Array Type="{x:Type x:String}">
          <x:String>Home</x:String>
          <x:String>Mobile</x:String>
          <x:String>Work</x:String>
          <x:String>Other</x:String>
        </x:Array>
      </Picker.ItemsSource>
    </Picker>
  </HorizontalStackLayout>
</FlexLayout>
```

ADVERTISERS INDEX

Advertisers Index

| | |
|---|--------|
| 1&1 Internet, Inc.
www.1and1.com | 7 |
| CODE Consulting
www.codemag.com/techhelp | 2, 57 |
| CODE Divisions
www.codemag.com | 75 |
| CODE Framework
www.codemag.com/framework | 49 |
| CODE Magazine
www.codemag.com/magazine | 37, 65 |
| CODE Staffing
www.codemag.com/staffing | 25 |
| dtSearch
www.dtSearch.com | 69 |
| LEAD Technologies
www.leadtools.com | 5 |

Advertising Sales:
Tammy Ferguson
832-717-4445 ext 26
tammy@codemag.com

This listing is provided as a courtesy to our readers and advertisers. The publisher assumes no responsibility for errors or omissions.

Display Images on a Button

The Save and Cancel buttons you added to the page use text to display what each button does. If you wish to use an image instead of text, change the Button control to an ImageButton control. Use the **Source** property to

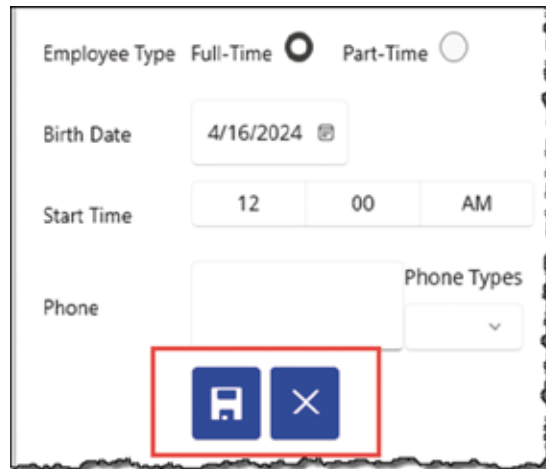


Figure 7: Use an ImageButton to display a graphic on a button instead of text.

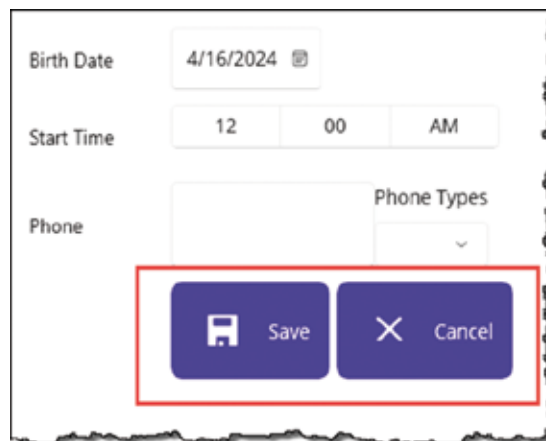


Figure 8: Place an image and text within a Button control.

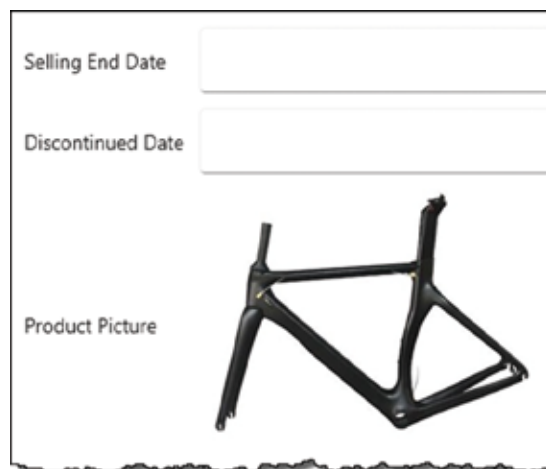


Figure 9: Display a picture using the Image control.

reference an image contained in the **Resources\Images** folder. When I'm working with just an image, I like to add a Tooltip for when a user hovers over the button on a Windows computer or when they do a long press on a mobile device. This is accomplished by setting the **ToolTipProperties.Text** property to the appropriate text.

Modify the Save and Cancel buttons in the last <HorizontalStackLayout> element on the User Detail page to become ImageButton controls. I have created **save.png** and **cancel.png** images to show a disk icon for save and the letter X for cancel, as shown in **Figure 7**. Either download the samples for this article or locate similar images by doing a Google search. The images should be 32 pixels by 32 pixels for the height and width. Place these images into the **Resources\Images** folder. Replace the last <HorizontalStackLayout> element in the grid with the following XAML:

```
<HorizontalStackLayout Grid.Row="11"
                        Grid.Column="1">
    <ImageButton Source="save.png"
                  ToolTipProperties.Text="Save Data"
                  Clicked="SaveButton_Clicked" />
    <ImageButton Source="cancel.png"
                  ToolTipProperties.Text="Cancel Changes" />
</HorizontalStackLayout>
```

The ImageButton control doesn't have any default styling, so open the **CommonStyles.xaml** file in the **Resources\Styles** folder in the **Common.Library.MAUI** project and add a new global style as shown in the following XAML:

```
<Style TargetType="ImageButton">
    <Setter Property="BackgroundColor"
            Value="Blue" />
</Style>
```

When dealing with images in a .NET MAUI application, make sure the name of the image is all lowercase, as mixed case image names don't work on all mobile devices. Underlines are acceptable but try not to use any other special characters as they may also not work on all operating systems.

Try It Out

Run the application and click on **Users > Navigate to Detail** to see the new ImageButton controls with the save and cancel images displayed, as shown in **Figure 7**.

Image and Text in a Single Button

What if you wish to have both an image and text (**Figure 8**) within a button? The Button control supports an **ImageSource** property that allows you to set the name of an image located within the **Resources/Images** folder. There's also a **ContentLayout** property to help you position where within the button you want the image placed. The valid values for this property are **Left**, **Top**, **Right**, **Bottom**. If you wish to add some spacing between the image and the text, place a comma and then the number of device-independent pixels you want. For example, set **ContentLayout="Left,40"** to position the image on the left, with 40 pixels between the image and the text.


```
<HorizontalStackLayout Grid.Row="11"
                        Grid.Column="1">
  <Button Text="Save"
    ImageSource="save.png"
    ToolTipProperties.Text="Save Data"
    ContentLayout="Left"
    Clicked="SaveButton_Clicked" />
  <Button Text="Cancel"
    ImageSource="cancel.png"
    ContentLayout="Left"
    ToolTipProperties.Text="Cancel Changes" />
</HorizontalStackLayout>
```

Try It Out

Run the application and click on **Users > Navigate to Detail** to see the buttons with both an image and text, as shown in **Figure 8**.

Image Control

To display a picture on a page where there's typically no interaction with that picture, use the **Image** control. The Image control has a **Source** property to which you set the name of a file located in the **Resources\Images** folder. The **Source** property may also access a URI or a stream to display the picture. As previously mentioned, make sure your picture file names are all lowercase and don't contain any special characters other than maybe an underscore. The **Aspect** property controls how the picture is scaled when displaying within the container.

Let's add a Label control to display "Product Picture". Then use the Image control to display a picture, as shown in **Figure 9**. Open the **Views\ProductDetailView.xaml** file and add an "Auto" to the **RowDefinitions** attribute of the Grid control. Locate the last **<HorizontalStackLayout>** starting tag at the bottom of the Grid and change the **Grid.Row** property to "14" instead of "13". Just before the **<HorizontalStackLayout>** starting tag you just modified add the following XAML that creates the Image control:

```
<Label Text="Product Picture"
  Grid.Row="13" />
<Image Grid.Row="13"
  Grid.Column="1"
  HorizontalOptions="Start"
  Aspect="Center"
  Source="bikeframe.jpg" />
```

Try It Out

Run the application and click on **Products** to see the new Image control, as shown in **Figure 9**.

Editor Control

To allow the user to enter a large amount of text, use the **Editor** control. Use the **AutoSize** property to tell the editor to automatically change the size of the control to accommodate the user input. By default, this property is set to false. Useful properties for this control are **IsSpellCheckEnabled**, **IsTextPredictionEnabled**, **MaxLength**, and **Placeholder**.

Let's add a Product Notes entry area to the product detail page and use the Editor control to allow a user to add multiple lines of text. Open the **Views\ProductDetail-**

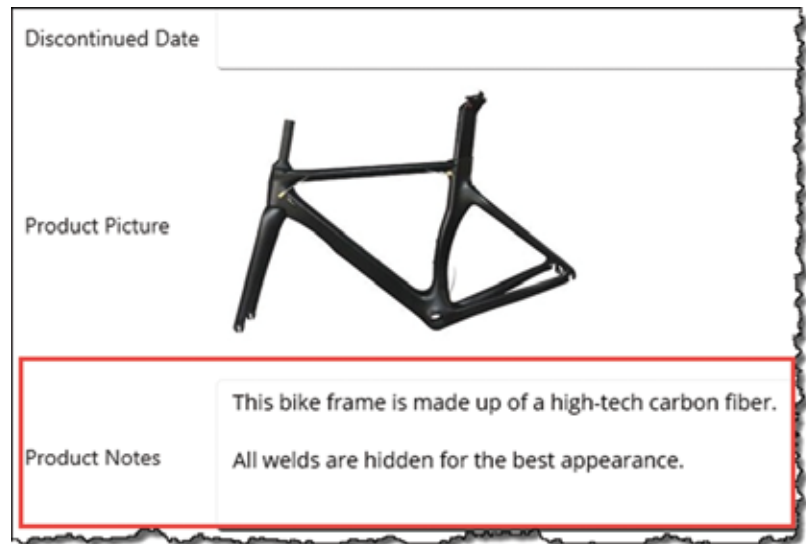


Figure 10: Use the Editor control to allow the user to enter multiple lines of text.



Figure 11: Use a Slider to allow the user to select a value between a minimum and a maximum.

View.xaml file and add an "Auto" to the **RowDefinitions** attribute of the Grid control. Locate the last **<HorizontalStackLayout>** starting tag at the bottom of the Grid and change the **Grid.Row** property to "15" instead of "14". Just before the **<HorizontalStackLayout>** starting tag you just modified, add the following XAML that creates the Image control:

```
<Label Text="Product Notes"
  Grid.Row="14" />
<Editor Grid.Row="14"
  Grid.Column="1"
  HeightRequest="100" />
```

Try It Out

Run the application and click on **Products** to see the new Editor control, as shown in **Figure 10**.

Binding a Slider to a Label Control

A **Slider** control displays a long horizontal rule along which you can drag a "thumb" to change the value (see **Figure 11**). Using the **Minimum** and **Maximum** property, you can control the range that's put into the **Value** property as the user drags the thumb. The Value property is a data type of double. If you wish to have the value change in whole number increments, you need to write a single line of C# code.

Let's replace the Weight Entry control on the product detail page to use a Slider control instead. Open the **Views\ProductDetailView.xaml** file and locate the Label and Entry control for Weight. Leave the Label control but replace the Entry control with the following XAML:

```
<VerticalStackLayout Grid.Row="7"
    Grid.Column="1">
    <Slider x:Name="weight"
        Value="1"
        Minimum="1"
        Maximum="100"
        ValueChanged="Weight_Changed" />
    <Label BindingContext="{x:Reference weight}"
        Text="{Binding Value}" />
</VerticalStackLayout>
```

Use data binding to see the value from the Slider appear in the Label as the thumb is dragged back and forth. In this code, you set the **Minimum** property to a value of one (1) and **Maximum** property to a value of 100. Set the initial value in the **Value** property to a one (1). Always set a starting value to avoid having to handle nulls in the code behind. In the Label control, set the **BindingContext** property to reference the "weight" Slider control and bind the **Text** property of the Label to the **Value** property in the Slider. As the thumb is dragged back and forth on the slider, the label displays the current number from the **Value** property through the magic of data binding.

After adding the XAML, position your cursor on the "Weight_Changed" in the ValueChanged attribute and press F12. This creates the event procedure to be called each time the user slides the thumb on the slider. Write the following code in this event procedure:

```
private void Weight_Changed(object sender,
    ValueChangedEventArgs e) {
    weight.Value = Math.Round(e.NewValue, 0);
}
```

The argument, **e**, received in this event procedure, contains both the **OldValue** and **NewValue** properties each time it's called. As mentioned, the **Value** property is of type double, so if you wish to use integer increments, simply apply the **Math.Round()** method to the **e.NewValue** property, passing zero (0) as the second parameter for the number of digits to use when rounding.



Figure 12: Use the stepper control to increment and decrement values.

Try It Out

Run the application and click on **Products** to see the new Slider control, as shown in **Figure 11**. Move the thumb back and forth to watch the new value appear in the label immediately below the slider.

Bind a Stepper to an Entry Control

The **Stepper** control allows you to increment or decrement a number, as shown in **Figure 12**. By binding a Stepper to an Entry control, you allow a user to either type in a number or click on the plus or minus sign on the Stepper. The **Value** property on a Stepper either gets or sets the current number. The **Minimum** and **Maximum** properties allow you to specify the range of values the user can move among. The **Increment** property lets you set how much to increase or decrease the **Value** property. The **Value** property is a double data type, so the Increment property can be set to almost any value you wish.

Let's bind two Stepper controls to both the Cost and Price Entry controls on the product detail page. Open the **Views\ProductDetailView.xaml** file and locate the <Entry> element below the <Label Text="Cost"> and replace it with the following XAML:

```
<HorizontalStackLayout Grid.Row="4"
    Grid.Column="1">
    <Entry Text="{Binding Value}"
        BindingContext="{x:Reference CostStepper}" />
    <Stepper x:Name="CostStepper"
        Value="5"
        Minimum="1"
        Maximum="9999"
        Increment="1" />
</HorizontalStackLayout>
```

Locate the <Entry> element below the <Label Text="Price"> and replace it with the following XAML:

```
<HorizontalStackLayout Grid.Row="5"
    Grid.Column="1">
    <Entry Text="{Binding Value}"
        BindingContext=
            "{x:Reference PriceStepper}" />
    <Stepper x:Name="PriceStepper"
        Value="10"
        Minimum="1"
        Maximum="9999"
        Increment="1" />
</HorizontalStackLayout>
```

Try It Out

Run the application and click on the **Products** menu. Increment and decrement each stepper to see the value change in the corresponding Entry controls, as shown in **Figure 12**.

Bind Two Steppers to One Another

Increase the Cost value to ten (10) and decrease the Price value to five (5). This should not be allowed, as you never want to sell something for less than it costs to make. Locate the <Stepper x:Name="CostStepper" ...> and modify it to look like the following XAML:

```
<Stepper x:Name="CostStepper"
    Value="5"
    Minimum="1"
    Maximum="{Binding Value}"
    BindingContext="{x:Reference PriceStepper}"
    Increment="1" />
```

Locate the **<Stepper x:Name="PriceStepper" ...>** and modify it to look like the following XAML:

```
<Stepper x:Name="PriceStepper"
    Value="10"
    Minimum="{Binding Value}"
    Maximum="9999"
    BindingContext="{x:Reference CostStepper}"
    Increment="1" />
```

The key to the code above is that you're binding the **Maximum** property on the cost stepper to the **Value** property of the price stepper. You then bind the **Minimum** property of the price stepper to the **Value** property of the price stepper. By binding these two stepper controls to one another, it solves the issue of being allowed to set the cost greater than the price.

Try It Out

Run the application and click on the **Products** menu. Increment and decrement each stepper to see the value change in the corresponding Entry controls. Notice that you can no longer make the price less than the cost, nor can the cost be greater than the price.

Disable One Control by Binding to IsEnabled on a CheckBox

A common business rule you encounter when programming is that you need to only make certain controls enabled, or visible, when another control, such as a RadioButton or CheckBox, is selected. Prior to data binding, this was accomplished using C# code. With data binding, you may now bind the **IsEnabled** or the **IsVisible** property on a control to the **IsChecked** property of another control.

Open the **Views\UserDetailView.xaml** file and locate the CheckBox under the **<Label Text="Flex Time?">**. Then add an **x:Name** attribute. You need this attribute so you can reference the CheckBox object with that name from another control.

```
<CheckBox x:Name="FlexTime" />
```

Locate the TimePicker and add an **IsEnabled** property to bind to the **IsChecked** property on the FlexTime CheckBox. Set the **BindingContext** attribute to refer to the FlexTime check box with the **x:Name** set to "FlexTime" as shown in the code snippet below.

```
<TimePicker Grid.Row="9"
    Grid.Column="1"
    BindingContext="{x:Reference FlexTime}"
    IsEnabled="{Binding IsChecked}"
    Time="08:00:00" />
```

Try It Out

Run the application and click on **Users > Navigate to Detail** and check the Flex Time check box to see the Start

Time TimePicker control become enabled. Uncheck the Flex Time check box to see that the Start Time becomes disabled.

Create a XAML Array Resource and Bind to Picker

In the last article, you created a Picker control and within that control, you created an array of phone types as the data for that picker. If you need to use that array on other pages, you must copy and paste the array from one page to another. This creates a maintenance nightmare, as you now have the same data in multiple places. Instead, let's move that data into a resource and bind the Phone Type Picker to that resource.

Open the **Resources\Styles\AppStyles.xaml** file and add the array of phone type strings as a resource. You need to provide a **key** to the array to be able to reference it from XAML pages, as shown in the code below:

```
<!-- Phone Types Resource -->
<:Array x:Key="phoneTypes"
    Type="{x:Type x:String}">
    <x:String>Home</x:String>
    <x:String>Mobile</x:String>
    <x:String>Work</x:String>
    <x:String>Other</x:String>
</x:Array>
```

Open the **Views\UserDetailView.xaml** file and locate the **<Picker>** and make it look like the following:

```
<Picker VerticalTextAlignment="Center"
    ItemsSource="{StaticResource phoneTypes}" />
```

Try It Out

Run the application and click on **Users > Navigate to Detail** to see the picker loaded with phone type data that's now coming from the global resource.

Bind Controls to Properties in a Class

Besides just binding one control to another, you may bind controls to the data you get from a C# class. For example, if you have a User class with properties named LoginId, FirstName, LastName, etc., you can easily bind those properties to the Entry, RadioButton, CheckBox, and other controls on the user detail page. Let's create the User class and learn how to bind the properties to the various controls.

I prefer to create all my "Entity" classes, such as the User class, in a separate class library project. Right mouse-click on the **Solution** and add a new **Class Library** project named **AdventureWorks.EntityLayer**. Delete the **Class1.cs** file as you don't need this file. Right mouse-click on the **AdventureWorks.EntityLayer** project and add a new folder named **EntityClasses**. Right mouse-click on the **EntityClasses** folder and add a new class named **User**. Replace the entire contents of this new file with the code shown in **Listing 3**. Right mouse-click on the **AdventureWorks.MAUI** project's **Dependencies** folder and add a Project Reference to the **AdventureWorks.EntityLayer** project.

Listing 3: Create a User class with properties to bind to the user detail page controls

```
namespace AdventureWorks.EntityLayer;

public class User {
    public int UserId { get; set; }
    public string LoginId { get; set; }
    = string.Empty;
    public string FirstName { get; set; }
    = string.Empty;
    public string LastName { get; set; }
    = string.Empty;
    public string Email { get; set; }
    = string.Empty;
    public string Password { get; set; }
    = string.Empty;
    public string Phone { get; set; }
    = string.Empty;
    public string PhoneType { get; set; }
    = string.Empty;
    public bool IsFullTime { get; set; }
    public bool IsEnrolledIn401k { get; set; }
    public bool IsEnrolledInHealthCare
    { get; set; }
    public bool IsEnrolledInHSA { get; set; }
    public bool IsEnrolledInFlexTime
    { get; set; }
    public bool IsEmployed { get; set; }
    public DateTime BirthDate { get; set; }
    = DateTime.Now.AddYears(-18);
    public TimeSpan StartTime { get; set; }
    = new TimeSpan(8, 0, 0);
}
```

Listing 4: Create an instance of the User class using XAML

```
<ContentPage.Resources>
    <vm:User x:Key="viewModel"
        LoginId="JohnSmith123"
        FirstName="John"
        LastName="Smith"
        Email="John@smith.com"
        Phone="615.222.2333"
        PhoneType="Mobile"
        IsFullTime="True"
        IsEnrolledIn401k="True"
        IsEnrolledInHealthCare="True"
        IsEnrolledInHSA="False"
        IsEnrolledInFlexTime="True"
        IsEmployed="True"
        BirthDate="10-03-1975"
        StartTime="08:00:00" />
</ContentPage.Resources>
```

Create a User Object Using XAML

Of course, you can write C# code to instantiate an instance of the User class, but you may also use XAML to create an instance. Open the **Views\UserDetailView.xaml** file and add an XML namespace that points to the new class library you just created.

```
xmlns:vm="clr-namespace:
    AdventureWorks.EntityLayer;
    assembly=AdventureWorks.EntityLayer"
```

Add an **x:DataType** attribute to the **<ContentPage>** element to make this page use compiled bindings. Compiled bindings improve the speed of data binding at runtime by resolving binding expressions at compile-time. In addition,

having this **x:DataType** attribute allows Visual Studio to read the properties of the class so IntelliSense can display those property names when you're assigning data binding to controls. Assign to the **x:DataType** the User class from the EntityLayer that's aliased as "vm", shown in the code below.

```
x:DataType="vm:User"
```

Create a **<ContentPage.Resources>** element just below the **<ContentPage>** starting tag and declare an instance of the User class within the XAML. You must assign a unique name to this instance using the **x:Key** attribute, as shown in **Listing 4**. Now that you have an instance of the User class created, you need to be able to bind each property to each control. Set a **BindingContext** on one of the parent controls to each of the individual controls on this page. You can either use the Border, the ScrollView, or the Grid. I like to use the outermost parent, so modify the Border control to have a **BindingContext** property that points to the **viewModel** resource, as shown in the following XAML:

```
<Border Style="{StaticResource Border.Page}"
    BindingContext="{StaticResource viewModel}">
```

Bind Entry Controls

First, let's bind each of the Entry controls on the page. Add a **Text** property to each Entry control if it's not already there. Within the **Text** property, use the **Binding** markup extension in the format **{Binding PROPERTY_NAME}** where you replace PROPERTY_NAME with the property name from the User class. In the following code snippet is a list of all the Entry controls and their corresponding properties:

```
<Entry Text="{Binding LoginId}" />
<Entry Text="{Binding FirstName}" />
<Entry Text="{Binding LastName}" />
<Entry Text="{Binding Email}" />
<Entry Text="{Binding Phone}" />
```

Notice that, as you type in this code, after you press the spacebar, a list of property names in the User class is displayed. If you don't get a list of property names, rebuild the solution. Continue down the page and map each Entry control to the appropriate property.

Bind Check Box Controls

Locate each **<CheckBox>** element and set each one's **IsChecked** property to the corresponding property in the User class, as shown in the following code snippet:

```
<CheckBox
    IsChecked="{Binding IsEnrolledIn401k}" />
<CheckBox
    IsChecked="{Binding IsEnrolledInFlexTime}" />
<CheckBox
    IsChecked="{Binding IsEnrolledInHealthCare}" />
<CheckBox
    IsChecked="{Binding IsEnrolledInHSA}" />
```

Bind the Switch Control

Locate the **<Switch>** control and set its **IsToggled** property to the **IsEmployed** property, as shown in the following snippet:

```
<Switch Grid.Row="6"
        Grid.Column="1"
        IsToggled="{Binding IsEmployed}" />
```

Bind Radio Button Controls

Locate the `RadioButton` control under the `<Label Text="Full-Time" />` element and set the **IsChecked** property to the following:

```
<RadioButton IsChecked="{Binding IsFullTime}"
             GroupName="EmployeeType" />
```

Don't set the **IsChecked** property on the Part Time `RadioButton`; you'll learn how to do that later.

Bind the Date Picker Control

Locate the `<DatePicker>` element and bind the **Date** property to the **BirthDate** property, as shown in the following XAML:

```
<DatePicker Grid.Row="8"
            Grid.Column="1"
            Date="{Binding BirthDate}" />
```

Bind the TimePicker Control

Locate the `<TimePicker>` control and remove the **IsEnabled** and the **BindingContext** attributes. Add the **Time** attribute and bind it to the **StartTime** property on the **User** class.

```
<TimePicker Grid.Row="9"
            Grid.Column="1"
            Time="{Binding StartTime}" />
```

Picker

Modify the **Picker** for the phone types and add the **SelectedItem** attribute to bind to the **PhoneType** property in the **User** class.

```
<Picker VerticalTextAlignment="Center"
        SelectedItem="{Binding PhoneType}"
        ItemsSource="{StaticResource phoneTypes}" />
```

Try It Out

Run the application and click on **Users > Navigate to Detail** to see the data from the **User** object appear in each of the controls.

Use a ValueConverter to Change a Boolean Value

In the **User** class, you have an **IsFullTime** property, but there's no **IsPartTime** property because if the **IsFullTime** property is set to false, it indicates that the user is a part-time employee. So, how do you get the **IsChecked** property on the Part Time `RadioButton` to be true when the **IsFullTime** property is set false?

As this is a very common scenario, Microsoft created an **IValueConverter** interface. This interface is implemented on a class to which you write code for both `Convert()` and `ConvertBack()` methods. The `Convert()` method takes in a value from a property and you then write code to convert that value into a different value. For example, you can take the false value from the **IsFullTime** property and change it to the inverse to return a true value so it can

Listing 5: Create a Convert class to transform a value into a different value

```
using System.Globalization;

namespace Common.Library.MAUI.Converters;

public class InvertedBoolConverter
    : IValueConverter {
    public object? Convert(object? value,
        Type targetType, object? parameter,
        CultureInfo culture) {
        if (value != null) {
            return !(bool)value;
        }
        else {
            return null;
        }
    }

    public object? ConvertBack(object? value,
        Type targetType, object? parameter,
        CultureInfo culture) {
        return null;
    }
}
```

be mapped to the **IsChecked** property on the Part Time `RadioButton`. Let's look at how to create a converter class to perform this transformation.

Right mouse-click on the **Common.Library.MAUI** project and add a new folder named **Converters**. Right mouse-click on the **Converters** folder and add a new class named **InvertedBoolConverter**. Replace the entire contents of this new file with the code shown in **Listing 5**.

Bind the RadioButton Control

Now that you've created the converter class, you may pass in the **IsFullTime** property to that class and have the value changed to the opposite of what it currently is. Open the **Views\UserDetailView.xaml** file. Add an XML namespace to reference the namespace in which the converter class resides.

```
xmlns:converters="clr-namespace:
    Common.Library.MAUI.Converters;
    assembly=Common.Library.MAUI"
```

Create an instance of the converter class within the `<ContentPage.Resources>` element, as shown in the following code:

```
<ContentPage.Resources>
    <converters:InvertedBoolConverter
        x:Key="invertedBoolean" />

    // REST OF THE XAML HERE
</ContentPage.Resources>
```

Locate the `Radio Button` under the `<Label Text="Part-Time" />` element and bind the **IsChecked** property to the **IsFullTime** property. On the **Binding** markup extension, set the **Converter** property to the **invertedBoolean** resource you created. Before the binding takes place, the value in the **IsFullTime** property is first passed to the **Converter** and the value returned is bound to the **IsChecked** property.

Getting the Sample Code

You can download the sample code for this article by visiting www.CODEMag.com under the issue and article, or by visiting www.pdsa.com/downloads. Select "Articles" from the Category drop-down. Then select "Exploring .NET MAUI: Data Entry Controls and Data Binding" from the Item drop-down.

```
<RadioButton
    IsChecked="{Binding IsFullTime,
        Converter={StaticResource invertedBoolean}}"
    GroupName="EmployeeType" />
```

Try It Out

Before you run the application, set the **IsFullTime** property in the XAML instance of the User object to a **false** value. Run the application and click on **Users > Navigate to Detail**. Notice that the Part Time radio button is now checked.

Bind the Time Picker Control

Let's now bind the **IsEnabled** binding on the TimePicker control to use the **IsFullTime** property in the User class. Of course, you only want the user to choose a start time if the **IsFullTime** property value is false, so use the converter class when binding the **IsEnabled** property on the TimePicker control. Locate the Flex Time CheckBox and remove the **x:Name="FlexTime"** attribute from that check box. Locate the TimePicker control and set the **Time** property to bind to the **StartTime** property on the User class. Set the **IsEnabled** property to bind to the **IsFullTime** property on the User class but invert the value in this property using the converter **invertedBoolean**, as shown in the following XAML:

```
<TimePicker Grid.Row="9"
    Grid.Column="1"
    IsEnabled="{Binding IsFullTime,
        Converter={StaticResource invertedBoolean}}"
    Time="{Binding StartTime}" />
```

Try It Out

Run the application and click on the **Users > Navigate to Detail** to view the user detail page. Notice that the Start Time picker is enabled because you previously set the **IsFullTime** property to a false value. Stop the application and change the **IsFullTime** property to the value "True" on the User object created in the Content.Resources element. Run the application and click on the **Users > Navigate to Detail** to view the user detail page. Notice that the Start Time picker is now disabled. However, if you click back and forth between the Full Time and Part Time radio buttons, the TimePicker control is not enabled or disabled during runtime. You'll learn how to do that later in this article.

Access the User Object in Code Behind

After creating the User object in XAML, you most likely want to access that object from code behind so you can modify properties on it. Open the **Views\UserDetailView.xaml.cs** file and add a using statement at the top of the file.

```
using AdventureWorks.EntityLayer;
```

Add a private read-only variable of the data type **User** to the **UserDetailView** class.

```
private readonly User _viewModel;
```

You created the instance of the User class in the **<ContentPage.Resources>** element and assigned it the key "viewModel". You can use the **Resources** property in the

class and index into that collection using the key "viewModel". Modify the constructor to retrieve the object created by XAML and assign it to the **_viewModel** variable.

```
public UserDetailView() {
    InitializeComponent();

    _viewModel = (User)this.Resources["viewModel"];
}
```

In the **Clicked** event procedure on the **Save** button, add the line of code shown below.

```
System.Diagnostics.Debugger.Break();
```

Try It Out

Run the application and click on **Users > Navigate to Detail**. Make some changes to some of the data in the controls and click on the **Save** button. Hover over the **_viewModel** variable and you should see the changes you made on the page show up in the properties of the variable.

Try to Change Data in User Object

As you just saw, the changes made on the page are reflected in the bound object. You'd assume that if you make a change to a property in the code-behind, the control would be updated as well. Let's try this out and see what happens. Open the **Views\UserDetailView.xaml.cs** file and add code in the constructor to change the **LoginId** property on the **_viewModel** variable to a different value as shown below:

```
public UserDetailView() {
    InitializeComponent();

    _viewModel = (User)this.Resources["viewModel"];
    _viewModel.LoginId = "ANewLoginId123";
}
```

Try It Out

Run the application and click on **Users > Navigate to Detail**. Notice that the value in the Login ID field DOES NOT reflect the value you put in the constructor; it displays the value set in the XAML.

Notify the UI of Changes to Objects

.NET MAUI controls know that when they change, they must put the value back into the properties they're bound to, but the reverse is not true. To inform controls of changes when you set a property in your object to a new value, you need to raise an event. .NET MAUI controls listen for the **PropertyChanged** event, which informs them that a change was made in the underlying object. When this event is raised, the name of the property that was changed is sent in the **event arguments** object. The control that's bound to the property re-reads the value in the object's property and updates the control, so the user sees the change. This event needs to be raised in all the setters on each property in your classes.

Let's create a base class that all your classes can inherit from. In this class, you're going to implement the **INotifyPropertyChanged** interface. This interface has one event

declaration, `PropertyChanged`, that needs to be declared, and which is ultimately called anytime one of your property values changes. Because this interface can be used in .NET MAUI, Blazor, WPF, WinUI 3, and many other types of applications, I recommend placing this new base class in a separate class library.

Right mouse-click on the **Solution** and select **Add > New Project...** from the menu. Select **Class Library** from the project template list. Set the Project Name to **Common.Library**. Delete the **Class1.cs** file, as this file isn't needed. Right mouse-click on the **Common.Library** project and add a new folder named **BaseClasses**. Right mouse-click on the **BaseClasses** folder and create a class named **CommonBase**. Replace the entire code in this new file with the code shown in **Listing 6**.

In the `CommonBase` class, you have a constructor that calls an `Init()` method. This method is marked as virtual so you can override it in your classes if you wish to initialize properties to a valid start value. Next, you see the declaration of the `PropertyChanged` event. The final method is named `RaisePropertyChanged()` and it's this method you pass the name of the property that raises the `PropertyChanged` event. Pass the property name to the new instance of the `PropertyChangedEventArgs` class, as it's this object that informs the UI which bound control to update.

Create an Entity Base Class

Eventually, you're going to create many kinds of classes for your applications. You'll create many data classes, view model classes, entity classes, etc. I recommend that you create base classes for each type of these classes. To that end, create a base class that's just going to be inherited from your entity classes. This class is inherited from the `CommonBase` class so it gets all its functionality as well as any other functionality you may add that's specific just for your entity classes. Right mouse-click on the **BaseClasses** folder in the **Common.Library** project and add a new class named **EntityBase**. Replace the entire contents of the new file with the following code.

```
namespace Common.Library;

public abstract class EntityBase : CommonBase {
}
```

Set Dependencies to the New Class Library

Right mouse-click on the **Dependencies** folder in the **AdventureWorks.EntityLayer** and add a project reference to the **Common.Library** project. Right mouse-click on the **AdventureWorks.MAUI** project's **Dependencies** folder and add a project dependency to the **Common.Library** project.

Inherit from the EntityBase Class

It's now time to inherit the `EntityBase` class from your User class. Open the **EntityClasses\User.cs** file and add a using statement at the top of the file.

```
using Common.Library;
```

Change the class declaration so the User class inherits from the `EntityBase` class.

```
public class User : EntityBase
```

Listing 6: Create a base class for all your entity classes to inherit from

```
using System.ComponentModel;

namespace Common.Library;

public abstract class CommonBase :
    INotifyPropertyChanged {
    #region Constructor
    protected CommonBase() {
        Init();
    }
    #endregion

    #region Init Method
    /// <summary>
    /// Initialize properties
    /// </summary>
    public virtual void Init() {
    }
    #endregion

    #region RaisePropertyChanged Method
    /// <summary>
    /// Event used to raise changes
    /// to any bound UI objects
    /// </summary>
    public event PropertyChangedEventHandler?
        PropertyChanged;

    public virtual void RaisePropertyChanged(
        string propertyName) {
        this.PropertyChanged?.Invoke(this, new
            PropertyChangedEventArgs(propertyName));
    }
    #endregion
}
```

To illustrate the use of the `RaisePropertyChanged()` method, just replace the one of the properties right now. You added code to change the **LoginId** property in the constructor of the `UserDetailView` class, so replace the simple **LoginId** property with the following code:

```
private string _LoginId = string.Empty;

public string LoginId {
    get { return _LoginId; }
    set {
        _LoginId = value;
        RaisePropertyChanged(nameof(LoginId));
    }
}
```

In the setter for this property after setting the private variable, `_LoginId`, call the `RaisePropertyChanged()` method passing in the string name of this property.

Try It Out

Run the application and click on **Users > Navigate to Detail**. You should now see that the **LoginId** value has changed to the value you typed into the constructor. The User object assigned to the **viewModel** variable in the XAML is created when the call to the `InitializeComponent()` method is invoked in the constructor. At that time, the Entry control bound to the **LoginId** property contains the value created in the XAML. When you change the **LoginId** property with the assignment `_viewModel.LoginId = "ANewLoginId123"`, the `RaisePropertyChanged()` event is raised with the event argument set to "LoginId" and thus the Entry control bound to the **LoginId** property

Listing 7: All property set methods should call the RaisePropertyChanged() method

```

using Common.Library;

namespace AdventureWorks.EntityLayer;

public class User : EntityBase {
    #region Private Variables
    private int _UserId;
    private string _LoginId = string.Empty;
    private string _FirstName = string.Empty;
    private string _LastName = string.Empty;
    private string _Email = string.Empty;
    private string _Password = string.Empty;
    private string _Phone = string.Empty;
    private string _PhoneType = string.Empty;
    private bool _IsFullTime;
    private bool _IsEnrolledIn401k;
    private bool _IsEnrolledInHealthCare;
    private bool _IsEnrolledInHSA;
    private bool _IsEnrolledInFlexTime;
    private bool _IsEmployed;
    private DateTime _BirthDate
        = DateTime.Now.AddYears(-18);
    private TimeSpan? _StartTime
        = new(8, 0, 0);
    #endregion

    #region Public Properties
    public int UserId {
        get { return _UserId; }
        set {
            _UserId = value;
            RaisePropertyChanged(nameof(UserId));
        }
    }

    public string LoginId {
        get { return _LoginId; }
        set {
            _LoginId = value;
            RaisePropertyChanged(nameof(LoginId));
        }
    }

    public string FirstName {
        get { return _FirstName; }
        set {
            _FirstName = value;
            RaisePropertyChanged(nameof(FirstName));
        }
    }

    public string LastName {
        get { return _LastName; }
        set {
            _LastName = value;
            RaisePropertyChanged(nameof(LastName));
        }
    }

    public string Email {
        get { return _Email; }
        set {
            _Email = value;
            RaisePropertyChanged(nameof(Email));
        }
    }

    public string Password {
        get { return _Password; }
        set {
            _Password = value;
            RaisePropertyChanged(nameof(Password));
        }
    }

    public string Phone {
        get { return _Phone; }
        set {
            _Phone = value;
            RaisePropertyChanged(nameof(Phone));
        }
    }

    public string PhoneType {
        get { return _PhoneType; }
        set {
            _PhoneType = value;
            RaisePropertyChanged(nameof(PhoneType));
        }
    }

    public bool IsFullTime {
        get { return _IsFullTime; }
        set {
            _IsFullTime = value;
            RaisePropertyChanged(nameof(IsFullTime));
        }
    }

    public bool IsEnrolledIn401k {
        get { return _IsEnrolledIn401k; }
        set {
            _IsEnrolledIn401k = value;
            RaisePropertyChanged(nameof(IsEnrolledIn401k));
        }
    }

    public bool IsEnrolledInHealthCare {
        get { return _IsEnrolledInHealthCare; }
        set {
            _IsEnrolledInHealthCare = value;
            RaisePropertyChanged(nameof(IsEnrolledInHealthCare));
        }
    }

    public bool IsEnrolledInHSA {
        get { return _IsEnrolledInHSA; }
        set {
            _IsEnrolledInHSA = value;
            RaisePropertyChanged(nameof(IsEnrolledInHSA));
        }
    }

    public bool IsEnrolledInFlexTime {
        get { return _IsEnrolledInFlexTime; }
        set {
            _IsEnrolledInFlexTime = value;
            RaisePropertyChanged(
                nameof(IsEnrolledInFlexTime));
        }
    }

    public bool IsEmployed {
        get { return _IsEmployed; }
        set {
            _IsEmployed = value;
            RaisePropertyChanged(nameof(IsEmployed));
        }
    }
    }

```

Listing 7: continued

```
public DateTime BirthDate {
    get { return _BirthDate; }
    set {
        _BirthDate = value;
        RaisePropertyChanged(nameof(BirthDate));
    }
}

public TimeSpan? StartTime {
    get { return _StartTime; }
    set {
        _StartTime = value;
        RaisePropertyChanged(nameof(StartTime));
    }
}

}

public string FullName {
    get { return FirstName + " " + LastName; }
}

public string LastNameFirstName {
    get { return LastName + ", " + FirstName; }
}
}
#endregion
```

receives this notification to re-read the `LoginId` property in the object. When that happens, the `Entry` control now reads the value "ANewLoginId123" from the property and updates the **Text** property on itself.

Update all Properties on the User Class

All the properties in the **User** class follow the same design pattern as the one you just created for the `LoginId` property. Open the `EntityClasses\User.cs` file and replace the entire contents with the code shown in **Listing 7**.

Use the `OnAppearing()` Method

I don't like placing code that modifies properties in the constructor of a class. Doing this may cause exceptions that can be very difficult to troubleshoot. Instead, I recommend any changes to properties be performed in the `OnAppearing()` event. Setting properties, or calling methods, in this event also has the benefit that if the page is removed the shell, but later navigated back to, this event is fired again. This is ideal if you want to refresh the data from your data store and have the updated values displayed in the controls.

Open the `Views\UserDetailView.xaml.cs` file and remove the line of code that changes the **LoginId** in the constructor. Add the following code to override the `OnAppearing()` event:

```
protected override void OnAppearing() {
    base.OnAppearing();

    _ViewModel.LoginId = "PeterPiper384";
    _ViewModel.FirstName = "Peter";
    _ViewModel.LastName = "Piper";
    _ViewModel.Email = "Peter@piperinc.com";
}
```

Try It Out

Run the application and click on **Users > Navigate to Detail** to see the changes in each control that are bound to the properties you modified in the `OnAppearing()` event.

Summary

In this article, you learned many ways for the user to provide input other than simple fill-in-the-blank `Entry` controls. You also saw how to bind one control to another to solve some basic business application scenarios without writing C# code. You took advantage of using data bind-

ing to bind the properties of a `User` object to controls on the page. If you want to make changes to an object in your code behind, you need to raise the `PropertyChanged` event to have that change reflected in the user interface. Coming up in the next article, you'll be introduced to the Model-View View Model (MVVM) and the Dependency Injection (DI) design patterns. Both of these design patterns make your applications reusable, maintainable, and testable. In addition, commanding eliminates code behind and moves that code down into your view model classes for even more reusability.

Paul D. Sheriff
CODE

SPONSORED SIDEBAR

Ready to Modernize a Legacy App?

Need advice on migrating yesterday's **legacy applications** to today's modern platforms? Take advantage of **CODE Consulting's** years of experience and contact us today to schedule a **FREE** consulting call to discuss your options.

**No strings.
No commitment.**

For more information:
www.codemag.com/consulting or email us at info@codemag.com.

First Rule of ARIA: Don't Use ARIA

As you expand your accessibility knowledge, you've probably heard the term ARIA a few times, maybe with an explanation, maybe not. Let's start there: ARIA (<https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA>) is a standard from the World Wide Web Consortium (W3C) (<https://www.w3.org/>) via the Web Accessibility Initiative (WAI) (<https://www.w3.org/WAI/>).



Ashleigh Lodge

ashleigh.lodge@gmail.com

Ashleigh is the Application Development Manager at Neovation Learning Solutions in Winnipeg, Manitoba, Canada, managing a team of over 20 developers, UX designers, and QA analysts. A graduate of the Computer/Analyst Programmer (CAP) program at Red River College, she has held various development and management roles in the tech industry for over 10 years.

A vocal advocate for accessibility and inclusive design, Ashleigh has spoken about accessibility at Prairie Dev Con Winnipeg and Regina, and at TedxWinnipeg.

In her free time, Ashleigh consumes a truly frightening amount of pop-culture media, including movies, TV shows, comic books, and novels. You can usually find her with Pokémon Go open on her phone, no matter where she is or what she's (supposed to be) doing.



ARIA is an acronym that stands for **Accessible Rich Internet Applications**, which is a bit of a mouthful, but it's important to understand what ARIA is for, and it's all right in the name. The first A is Accessible, which makes sense—we all want our work to be accessible. The last three letters in the acronym should be taken as a group, referring to Rich Internet Applications (https://en.wikipedia.org/wiki/Rich_Internet_Application), which, as per Wikipedia, are "web application[s] that ha[ve] many of the characteristics of a desktop application... and may allow the user interactive features such as drag and drop, background menu, WYSIWYG editing, etc." Over the history of the internet, we've seen the web evolve from static sites, to simple applications in the browser, to the complex, distributed web applications we have now. First introduced in 2015, ARIA has always been meant for complex web applications, not simple pages or sites like blogs.

Another way to think of ARIA is as a polyfill for HTML semantics, as per Sara Soueidan (<https://www.sarasoueidan.com/>), a freelance developer and designer who teaches accessibility workshops around the world. As you may know, a polyfill ([https://en.wikipedia.org/wiki/Polyfill_\(programming\)](https://en.wikipedia.org/wiki/Polyfill_(programming))) is "code that implements a feature on browsers that don't support the feature." HTML semantics are crucial for accessibility because they provide context and meaning to our code (refer to my previous articles for more information: POURing Over Your Website at <https://www.codemag.com/Article/1911091/POURING-Over-Your-Website-An-Introduction-to-Digital-Accessibility> and Easy Accessibility Wins at <https://www.codemag.com/Article/2312041/Easy-Accessibility-Wins-Better-Accessibility-in-Five-Minutes-or-Less>) and should always be the first tool you reach for. But what about when those semantics simply don't exist? There's no such thing as an HTML tab UI element, so how do you build one and ensure that it's accessible? That's where ARIA comes in: You use ARIA to provide the missing meaning and context, so someone using assistive technology knows that tab 1 is linked to content section A, and tab 2 is content section B, etc.

Context

Context is crucial to accessibility, and native HTML elements provide this context by default. For example, if you take an HTML button, you know that it's a specific object on the page, separate and distinct from other objects. And you know what a button's role is: to be clicked! A button element (<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/button>) has focus and click events, which are sent to the browser and the accessibility tree (https://developer.mozilla.org/en-US/docs/Glossary/Accessibility_tree); a button can be disabled and a button can have a name, among other features.

On the other hand, you have elements like divs (<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/div>)

(or spans) that don't provide any of this context—because they're not meant to. A div is a generic element, and although you can make visual changes to have it appear to be a button, it won't be clickable without JavaScript and it won't be accessible without ARIA.

ARIA can provide context to an element, such as a div, in three ways. ARIA can add a **role** onto a generic element (i.e., a div becomes a button) or override the role that is provided to an element by default. ARIA can indicate the **relationship** between two elements (such as with `aria-labelledby`). And ARIA can provide information as to the element's **state** (i.e., a div being used as a checkbox that's either checked or unchecked).

Semantic elements, such as buttons, automatically provide information on role, relationship, and state to the browser and accessibility tree, but for custom elements, it's up to you to provide that information (and update it when necessary). If you have text inside a p tag and text inside a div, only one of those provides additional context and meaning to assistive technology. (It's the p tag!)

Role, Relationship, and State

There are three foundational elements to ARIA: role, relationship, and state, and each one is essential to using ARIA correctly.

Role

Think of ARIA roles like HTML elements, where you're declaring **what** an element is. A button has a role of button, a heading has a role of heading, and a tab would have a role of tab (but would be a div or list base element as there is no native tab element in HTML).

An easy trap to fall into is assuming that visual context and meaning—like styling a div to look like a tab—also provides accessible context and meaning, but of course it doesn't.

There are six categories of ARIA roles available to you.

Abstract roles (https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Roles#6._abstract_roles) can be confusing if you've seen them without context. The introduction from the ARIA spec says: "The following roles are used to support the WAI-ARIA role taxonomy for the purpose of defining general role concepts," which clears things right up. All you need to know is that Abstract roles are not for content at all; they're purely used by browsers and should not be used by developers. Phew!

Thankfully, **widget** roles (https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Roles#2._widget_roles) are much clearer in both definition and use. Widgets are interactive standalone or composite elements.

Some examples include our old friend the tab element or a tree navigation element. Composite widgets are containers that wrap and manage other widgets inside them. For example, a tablist widget is a composite widget that includes multiple tab widgets as children. A tree composite widget includes treeitem widget child elements.

Depending on the list of ARIA widget roles you find, you may see roles that overlap with HTML elements, such as button, checkbox, link, and radio. This is either because the ARIA role pre-dates the HTML element or because there are situations where the HTML element cannot or should not be used. When everything else is equal, always choose the semantic HTML element over a corresponding ARIA role.

Although widgets are interactive elements, **document structures** (https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Roles#1._document_structure_roles) are (usually) not. As you'd expect, document structure roles apply to sections of your document or page. As with widgets, many previously common ARIA document structure roles should no longer be used, as they've since been created as semantic HTML elements. This includes roles such as article, figure, and table.

One interesting thing to note is that separator is both a widget and a document structure role. Any guesses as to why? It comes down to interactivity: If the separator is focusable, it's a widget. And if it's not focusable, it's a document structure (and you should be using the hr element instead).

Landmark roles (https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Roles#3._landmark_roles) are exactly what they sound like: roles that are used for navigation. As you know, ARIA is about assistive technology, and one of the most common types of assistive technology is a screen reader. Although a sighted user can use visual cues to see a banner, article, and form on a page, an assistive technology user may not be able to do so. And, if you've created some divs on your page to encompass a banner and an article, although you might think the job is done, remember that divs carry no semantic meaning, so you're not providing the context you think you are.

Giving your structural divs the appropriate role allows assistive technology users to navigate the page much like their non-assistive technology using counterparts: by narrowing in on the banner to see information on the next event, by quickly skimming through multiple articles to find the one they're interested in, and to locate the search bar to find contact information.

Landmark roles give developers the ability to mimic the semantic meaning provided by the visual appearance of elements, which, in turn, allows assistive technology users to distinguish between multiple generic divs quickly and easily.

If you have content that changes dynamically—say a success or failure message after the user performs an action—you'll want to use **live regions** (https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Roles#4._live_region_roles).

The figure shows a web form titled "Signup". It contains four input fields arranged in two columns. The left column has "Given Name:" and "Email Address:" labels above their respective input boxes. The right column has "Surname:" and "Password:" labels above their respective input boxes. Below the input fields, there are two buttons: "Cancel" and "Save".

Figure 1: A simple signup form with Cancel and Save buttons under the input fields.

The figure shows the same "Signup" form as in Figure 1, but with an additional message at the bottom. The message is enclosed in a box with a black border and contains a large exclamation mark icon followed by the text: "Your account has been created. Please check your email for access information." The "Cancel" and "Save" buttons are still present above this message.

Figure 2: A simple signup form with a success message at the bottom of the page, under the input fields and Cancel and Save buttons.

Look at **Figure 1** and imagine you're navigating this form, filling out each field, and then hitting the Save button once you're done. So far so good right?

Once you've clicked on Save and the request has been processed, the form updates to look like **Figure 2**, with a success message at the very bottom of the page, under the Save and Cancel buttons.

If you're a visual user, finding a message or alert on a page is usually simple—they tend to be positioned either above or below the control you just interacted with, and provide information so you know whether the action was successful (or not). But what if you're a screen reader user and don't primarily navigate by looking at the screen? A screen reader user will likely still be focused (in the focus state sense) on the Save button, which provides no indi-

cation that a message has appeared. In another situation, a visually impaired user may have increased the zoom level on the page, pushing the alert past the bottom of their visible screen.

Maybe you're an experienced screen reader user and are expecting a message to appear somewhere. But where is it? Depending on the preferences and standards of the website or application, the message could be above the form, or below, like in my example. It could even be off to one side or another—it's completely dependent on the decisions made by the developers and designers, and nearly impossible for someone to guess correctly the first time they encounter a form.

Try to put yourself in the position of a screen reader user encountering a form on an unfamiliar website for the first time. Maybe most sites you typically use have messages above the form, so you navigate using your screen reader all the way back up through the form—my sample form here is very short and simple, but many forms aren't, so this could be very time consuming. And then the message isn't at the top of the form on this website, so you navigate all the way back down through the form—going through every single field for the third time—to see if the message was underneath the buttons instead. Even worse, what if these success and error messages dismiss themselves, automatically disappearing after a set period of time? A screen reader user may never actually be able to find and access any type of success or failure message unless live regions are implemented.

Typically, an alert message is implemented within a div, which, as you know, has no semantic meaning on its own. This div is then given a live region role (alert, log, marquee, status, or timer) to ensure that assistive technology can pick up on the information contained within it.

In the sample form in **Figure 2**, you'd want to use the alert role, which immediately announces the text within the div so the user is aware that the page has changed in some way. If you had a site with a countdown of some type (perhaps an online quiz with a time limit, or purchasing concert tickets where the process has to be finished within a specific timeframe), the timer role would be more appropriate. The timer role doesn't automatically announce all changes because that would be intensely annoying, but it does make it possible to navigate directly to the element to hear the time left, among other features.

Finally, there are **window** roles (https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Roles#5._window_roles), which are used for anything that pops up, or acts as a window within the browser (like dialog boxes). There are only two window roles, `alertdialog` and `dialog`, and their only difference is that `alertdialog` is modal and `dialog` is (usually) non-modal. Typically, `alertdialog` is used when the user's workflow is being disrupted and a response is required, while `dialog` is less interruptive.

These roles are simple to use, but for modal dialogs in particular, there are other accessibility requirements that must be considered. For example, in situations where a response is required, it's essential that the user is "trapped" within the modal and prevented from accessing the underlying page (called a focus trap). Think about a session

timeout modal dialog window: The user must log back into the application to continue using it, but if the user isn't first alerted to the presence of the dialog and then trapped within it, they may be able to continue to work "under" the modal and not realize that nothing they do is being saved because their session has expired.

Relationship

The next element of ARIA is relationship, which admittedly is a personal term. The ARIA spec combines properties and state together, but I prefer to separate them to better understand the details of how ARIA is implemented. In the ARIA spec, properties are defined as how two or more elements are connected, which, to me, means relationships.

Relationships are generally straightforward. For example, there's `aria-owns`, which creates a parent/child relationship between elements, like you might find in a menu. For example, you might have Contact Us as a top-level menu item, and then options for phone, email, and physical address as child items underneath. Using `aria-owns`, you can create these relationships that are then communicated to assistive technology users in the same way a visual menu structure would be.

You may be familiar with `aria-describedby` and `aria-labelledby`, which can be used to link descriptions and labels to elements such as charts and images. A label is usually short and to the point, where a description is more detailed and verbose. If you've included a chart on your site, having a description of the axes, the data points, the labels etc., is necessary for the chart to be accessible to all users.

If you have an accordion with a button to expand and collapse the section, you may want to use `aria-controls` to indicate the relationship between the button and the section that can be expanded or collapsed. The button has an `aria-controls` attribute that references the ID of the associated section, again creating a relationship between seemingly unrelated items.

State

At last, we have state. As mentioned earlier, I see more of a distinction between state and properties (or relationships) than the ARIA spec does, and another point of difference is that states are dynamic, while relationships don't (usually) change after they're set.

ARIA states indicate what's currently happening with the element, but also, what could happen with or to it in the future. For example, if you think about a checkbox that's enabled (checked), you know that the option is currently ON and you can then infer that it can also be turned OFF.

Some examples of ARIA state include:

- `aria-busy`
- `aria-checked`
- `aria-disabled`
- `aria-expanded`
- `aria-hidden`
- `aria-invalid`
- `aria-pressed`
- `aria-selected`

As you can see, these are all dynamic: An element is either checked or not, expanded or not, hidden or not. A lot of these are very similar in use: aria-checked, aria-pressed, and aria-selected are all different versions of the same functionality, and you chose which one to use based on the type of control you're creating (checkbox, toggle button, and dropdown list/combo box respectively).

Some of these are less commonly used than others—for example, aria-busy is specific to live regions and indicates that the region is currently being updated (like a constantly scrolling log, for example).

Role, relationship, and state are the fundamentals of ARIA. If you're going to use ARIA (and that's a big if), you need to consider the roles, relationships, and states for all the non-standard/non-semantic elements you're creating or using. And that question, whether you're going to use it? That's a big one.

ARIA Rules

The thing about ARIA is that it's an amazing tool that you should never have to use. There **should** be standard and semantic HTML elements for all the functionality you want to include on your websites and apps and games. But the standardization process works slowly and tech moves quickly, so there are always going to be gaps between the proper and correct way of doing something and what's actually happening in the real world. This is where ARIA comes in, at least for accessibility.

The title of this article is a bit tongue in cheek, but it's also very serious. There are five rules in the official ARIA specification document, and the first one is:

"If you can use a native HTML element or attribute with the semantics and behavior you require already built in, instead of repurposing an element and adding an ARIA role, state, or property to make it accessible, then do so."

Which, fancy specification language aside, is really straightforward. If you need a button on a page, use the button element, not a div. If you need a link on the page, use the link element, not a div. If you need a select list on the page, say it with me, use the select element, not a div. Of course, there are exceptions to every rule...

Don't Do It*

You should not use ARIA if a semantic HTML element exists for your needs—except if the element is available but not fully implemented (i.e., browser support) or if it hasn't been implemented in an accessible way.

Figure 3 shows a snippet of the support for various types of HTML5 controls in the major browsers (Chrome, Edge,

Firefox, Internet Explorer, and Safari, from left to right) from html5accessibility.com. A green checkmark indicates that the control is implemented and fully accessible; a red X indicates that it's not implemented in an accessible way; an orange circle shows partial support; and a grey dash is not applicable.

In **Figure 3**, you can see that Internet Explorer doesn't fully support the number and range inputs, Firefox only partially supports the color input (Safari doesn't support it at all), and datalist, progress, and meter are also partially supported in some of the browsers.

To reiterate, this doesn't mean that you can't use those HTML5 controls in your website, but you do need to be aware that they haven't been fully and accessibly implemented in all the browsers that you may support for your users. If you visit the HTML5 Accessibility page and see an unimplemented or partially implemented element, that's a good time to reach for ARIA to close the gaps.

The second exception when it's appropriate to use ARIA is when you need an element that simply doesn't exist in HTML, like a tab interface. There's literally nothing else you can do, so you must use ARIA in those situations.

Finally, the ARIA spec states a third exception: "if the visual design constraints rule out the use of a particular native element, because the element cannot be styled as required." And I get it. We've all had a boss or stakeholder breathing down our neck about the look and feel of our sites and applications, and shouldn't that button be a slightly different shade of blue and what if it had rounded corners (but not too rounded), and and and. If you have a 100% must-do requirement to style an element in a certain way, and it's not possible with the built-in HTML element, you've got to do what you've got to do. But I do think that this exception offers people a bit too easy of a way out and everyone should really think about what they're doing before they drop a native element and reach for something custom.

We're in an age of massive and blazingly fast technological advancement and every single browser and device and app is doing things just a tiny bit differently. Frankly, there's no such thing as an experience that's 100% the same for all users, everywhere, in every situation. If you've got even the smallest bit of wiggle room to push back just a little about the drop-down in Firefox looking slightly different than in Chrome, do your best to advocate for a standardized control over a custom one.

Don't Do That Either

Alright, so I've thoroughly covered the first rule and its exceptions. The second rule is "Don't change native semantics, unless you really [really, really] have to."

The key aspect of this rule is around combining elements and how you need to have a solid understanding of what you're trying to do—and why—before you go putting ARIA roles onto everything.

Say you're setting up a tab UI element. You're, of course, thinking about accessibility from the beginning, so you want to make sure that each tab is a heading for better organization, discoverability, and navigation.

ELEMENT	CRITERIA	Chrome	Edge	Firefox	Safari	Opera
+ number input	Accessibly supported	✓	✓	✓	○	✓
+ range input	Accessibly supported	✓	✓	✓	○	✓
+ color input	Accessibly supported	✓	✓	○	—	✗
+ datalist	Accessibly supported	✓	✓	✓	○	✓
+ output	Accessibly supported	✓	✓	✓	—	✓
+ progress	Accessibly supported	✓	✓	✓	○	✓
+ meter	Accessibly supported	○	✓	○	—	✓
+ details	Accessibly supported	✓	✓	✓	—	✓
+ summary	Accessibly supported	✓	✓	✓	—	✓
+ dialog	Accessibly supported	✓	✓	—	—	—

Figure 3: A screenshot from html5accessibility.com showing the browser support for various types of controls (number, range, and color inputs, datalist, output, progress, meter, details, summary and dialog)

You also know that there's no native HTML element for tabs, so you'll definitely need to use ARIA. With those two points in mind, you write this code:

```
<h2 role="tab">Tab 1</h2>
<h2 role="tab">Tab 2</h2>
```

And that's it, you've got headings that are also tabs, done! Well, not so fast. What you've actually done here is taken the semantic meaning of the heading elements and **overridden** it. You no longer have the headings you were using for organization, discoverability, and navigation because those headings are now tabs as far as the accessibility tree is concerned.

A fundamental piece of ARIA is that no piece or aspect of ARIA affects the visual display of an element, or the use of the element via standard means (i.e., when not using assistive technology). These tabs are going to look like headings on the screen, and you'll be able to target them in CSS or JavaScript as headings. The only effect of adding the **role="tab"** piece is to make a screen reader or other assistive device announce these elements as tabs, not headings. Setting a role explicitly always overrides any existing roles or semantic meaning present on an element; you cannot add to what is already there, only replace it.

In a shocking twist, what's actually needed here is a couple of divs (or spans), not headings.

```
<div role="tab">
  <h2>Tab 1</h2>
</div>
<div role="tab">
  <h2>Tab 1</h2>
</div>
```

Now you have an element that started without any semantic meaning (the divs) that have been given meaning, via ARIA. You also still have your standard headings, with their role coming from the fact that they're semantic tags with inherent meaning. Someone using assistive technology on a page with this code will know that there's a tab element due to the role on the div, and will also still be able to navigate headings in the way they're most comfortable with (i.e., many screen readers have a hotkey that allows users to jump from heading to heading, like a table of contents).

Keyboards Exist

The third rule of ARIA reminds you that keyboards exist as more than an input device and that some people use them to navigate instead of a mouse. The official rule is: "All interactive ARIA controls must be usable with the keyboard." There are, of course, additional details, but what it boils down to is that if you can do it with a mouse, you must be able to do the same thing (or an equivalent action) with the keyboard. And if there are standard keys or key combinations (think CTRL+S to save) for functionality, you must make sure those work as well.

This rule specifically calls out keyboards and that's because most assistive devices used to navigate mimic keyboard functionality. Most of us don't have a sip and puff device (<https://en.wikipedia.org/wiki/Sip-and-puff>) available to test with, but we all have a keyboard. ARIA, of course, refers to the accessibility of specific controls, but more broadly, your entire site and app need to be navigable via keyboard, even if there are few interactive controls.

The most common issues with keyboard navigation are the result of using a non-semantic element, such as a div, to create a custom control, like a button or drop-down. The developer (or designer) either doesn't know, or forgets to implement features like focus (by default, divs are not interactive and thus not focusable), or activation via keyboard instead of clicking/tapping (using the Enter/Return keys or space bar, which triggers the keydown event, instead of the click event).

Focus

The next ARIA rule is about ensuring that focusable elements aren't disabled or hidden from assistive technology when using ARIA. The rule states: "Do not use **role="presentation"** or **aria-hidden=true** on a focusable element."

The presentation role is interesting: If you have absolutely no choice but to use tables for your layout (in an email, for example), by adding the role of presentation to the table itself, you're indicating to assistive technology that the table doesn't contain data and should be ignored. When you apply **role="presentation"** to an element, you're functionally removing all semantics from that element, as far as the accessibility tree is concerned.

If you investigated the different types of ARIA roles mentioned previously, you may have noticed the **role="none"** option, which was introduced because developers didn't understand that **role="presentation"** means no semantics. Browser support for **role="none"** isn't very robust, so you should stick to using presentation instead. Living standards are fun!

Back to the fourth rule and **aria-hidden**, which works almost the same way as **role="presentation"**. If **aria-hidden** is enabled (**aria-hidden="true"**), the element is hidden from the accessibility tree in the browser. Again, I'm only talking about the accessibility tree here, not anything visual. If you're using **role="presentation"** on a table or **aria-hidden** on a button, the table borders are still going to be visible unless you've removed them with CSS, and the button is still going to be present on the page and clickable. Let's look at some examples.

First, here's a table with a presentation role:

```
<table role="presentation">
  <tr>
    <td>Row 1, Col 1</td>
  </tr>
  <tr>
    <td>
      <button>My Button</button>
    </td>
  </tr>
</table>
```

Normally, assistive technology announces a table as a table, and if it's coded properly, the assistive technology is aware of the table headers and footers as well as row and column headers. As the user navigates through the contents of the table, each cell is announced individually, along with the row and cell headers for context.

By adding **role="presentation"** in this example (or **role="none"**), you've told the browser's accessibility tree that this table isn't really a table so it doesn't need to announce any of the standard table elements. Instead, the screen reader navigates as if the table wasn't even there, simply announcing elements within the table in the order they appear in the code—which, in this case, is the text in the first row and the button in the second row.

The presentation (and none) role effectively absorbs and negates any semantics that are related to the element they're used on, so in this case, anything to do with a table (thead, tfoot, tr, td, colspan, etc.) is removed when the accessibility tree is generated. This only affects the element it's used on (the table in the example), so any interactive elements inside the table are announced and work as expected—in this case, My Button in the second row. The first column is announced as and continues to function as a button. I cannot emphasize enough that ARIA has **no visual effect**. If you forget to remove the table's borders, visual users are still going to know it's a table, even if they're also using a screen reader and it's not **announced** as a table (which would be very confusing).

Next, an **aria-hidden** example:

```
<button aria-hidden="true">Spoon</button>
<div aria-hidden="true">
  <button>Fork</button>
</div>
```

Here's a button that uses **aria-hidden** and a button inside a div that uses **aria-hidden**. Both of these have the same result: As far as assistive technology goes, there's no spoon (and no fork). Visually, of course, both buttons appear, but any assistive technology **will not** be able to interact with either button. Unlike **role="presentation"**, **aria-hidden** is inherited by all child elements, making both buttons inaccessible, even though the second one is hidden indirectly.

Using **aria-hidden** is a bit of a nuclear option, but there are situations where it's appropriate. The general rule is to use **aria-hidden** when the content is decorative, such as a background image, or if it's duplicated, like an icon in a button that also has text. What you don't want to do with **aria-hidden** is to use it "just in case." If you have content that needs to be hidden from everyone, **display: none** or **visibility: hidden** removes those elements visually and from the accessibility tree. And as you know, you should always take advantage of behavior that's provided by default rather than creating custom functionality.

Names Matter

The last of the ARIA rules is that "[a]ll interactive elements must have an accessible name." If you're using the HTML label element (and you've set it up properly), then

congratulations, you've got an accessible name for your element!

There are two ways to use the label element to properly and accessibly label your fields. The first is to use the **for** attribute on the label and link it to the correct input field:

```
<label for="userName">User Name:</label>
<input type="text" id="userName" />
```

Visually, you'll still need to place the label in an appropriate location but any assistive technology knows about the relationship between these two elements automatically because the ID of the input field (userName) has been used in the label's **for** attribute to connect these elements.

The second way to use a label is to wrap it around the input field:

```
<label>
  User Name:
  <input type="text" />
</label>
```

However, you can only use this option if the inner element (in this example, the input field) is labelable (https://developer.mozilla.org/en-US/docs/Web/HTML/Content_categories#labelable) as per the HTML specification.

If, instead, you had a div inside the label instead of an input field:

```
<label>
  User Name:
  <div>This is a non-labelable element</div>
</label>
```

Then you'd have a visual label and name but not an accessible one. This is because divs aren't labelable, so the accessibility tree cannot create a relationship between the elements, even though the structure is the same as the previous example with the input field.

While we're talking about labelable elements, what are they?

- button
- input (except hidden inputs)
- meter
- output
- progress
- select
- textarea

That's basically a list of anything you might want to put a label on, of course!

And if, for some reason, you can't use the label element, there are **aria-label** and **aria-labelledby** to fall back on:

```
<input type="text" aria-label="User Name" />
<button aria-label="Save"></button>
```

Like with the label element, if an element is labelable, you can use **aria-label** directly on the element (which is

a nice little tautology). Again, these are not visual labels, just accessible ones. These would pass accessibility checks, but they might not be very usable regardless. What if the icon in the button isn't clear? What if the input field is using a placeholder label?

- What do you do if you have an unlabelable element that, for some reason, needs a label/accessible name? In that case, you'd reach for **aria-labelledby** to create a relationship between two elements:

```
<div id="userName">User Name:</div>
<span aria-labelledby="userName"></span>
```

In this example, you have a div acting as the label, which is then linked to the span using **aria-labelledby**. Assuming that your visuals make sense, this span now has a visible label and an accessible name. But just because you can doesn't mean you should. What's the purpose of that span and why can't it be a standard input field? Always start with the standard HTML elements and only add in ARIA if absolutely necessary.

Just because you can doesn't mean you should.

Why?

I was always a bit scared of ARIA when I first started out with accessibility. It seemed like this huge, looming monster just over my shoulder and I wasn't sure if I should be using it, or if I was using it properly and actually making my sites more accessible or making them worse instead.

Now that I'm more comfortable with ARIA, it (thankfully) doesn't seem that bad, and while yes, it can be complex and somewhat opaque, the rules are straightforward and I have more confidence in what I'm seeing and doing. I hope you've gained a similar level of comfort from this article! But to wrap things up, here's one more interesting fact related to ARIA.

There's an organization called Web Accessibility in Mind (WebAIM) (<https://webaim.org/>) that does an annual automated accessibility evaluation (<https://webaim.org/projects/million/>) of the top one million most visited websites. In February 2024, they found that 75% of the top one million most visited sites were using ARIA in some way (excluding ARIA landmarks). This number has grown in leaps and bounds over the last five years of the survey, starting at 60% in 2019 and increasing nearly every year since.

The really interesting bit, though, is that the pages in the survey with ARIA had, on average, 34.2% **more** errors than pages without ARIA. Again, if a page was using ARIA in some way, users experienced 15 additional errors compared to a similar page without ARIA.

Of course, correlation is not causation. A likely explanation is that pages with ARIA are probably more complex,

which is why they were using ARIA in the first place, which also makes errors more likely. But ARIA is a standard for making rich internet applications accessible, so it seems to me like there's a pretty large disconnect between the intent and implementation of ARIA in the real world.

A project called **Higher Ed in 4K** (<https://blog.pope.tech/2023/11/01/higher-ed-in-4k-introduction/>) did an accessibility analysis of up to 100 pages from the website of each college and university in the United States in 2019, with a follow up in 2020. One school in that survey had an average of 5,552 ARIA elements **per page** (<https://blog.pope.tech/2019/11/27/higher-ed-in-4k-2019/#aria>), with over 500,000 ARIA elements over the 96 pages that were sampled. Another school had more ARIA attributes on their pages than actual elements! As the report itself states, "this is a reminder that ARIA doesn't improve accessibility unless done correctly."

It's almost impressive, in a scary kind of way. How did this even happen? Turns out it was a lot of tabindex, ARIA popups, and ARIA expanded, most of which you can safely assume weren't needed at all or could have been replaced with semantic HTML.

I'm Canadian, so I'm not very familiar with American colleges and universities, but based on my experience with Canadian versions of such sites, I cannot imagine that they were all such "rich internet applications" that they absolutely needed that much ARIA (especially when keeping in mind that these types of automated scans can only hit publicly available pages, so these tests wouldn't have had access to any student or instructor areas at all).

I wanted to highlight these studies because they're real-world examples of why we should always start from the position of not using ARIA. Most of the time, there will be a more simple, more semantic, more accessible way to code your site than trying to add in ARIA. And if there isn't, at least you've taken the time to really consider the necessity of ARIA, and probably won't end up with over five thousand duplicate and/or non-functional ARIA attributes on your website. Probably.

Validation

I haven't touched on it much, but you'll always want to test that you've used ARIA in an appropriate way, and there are a number of tools available to help you do so.

The **ARIA Validator** by Rick Brown (<https://chromewebstore.google.com/detail/aria-validator/oiggghlanfjgnkcndchmnlmaojahnjoc?pli=1>) is a Chrome-only extension that will validate the ARIA you've used on your site against the spec and let you know about any issues. As with all automated accessibility checking and validation, this tool cannot find everything and the best way to ensure that you've written an accessible website, app, or game is to test it with a variety of disabled users.

The **Visual ARIA bookmarklet** (<https://whatsock.github.io/visual-aria/github-bookmarklet/visual-aria.htm>) from Apex 4K is a similar tool to the ARIA Validator above, allowing you to check the ARIA used against the spec, but it can be easy to install because it's a simple JavaS-

cript bookmarklet instead of an extension. Like with other types of accessibility checking, it's good to use more than one tool because they usually have their own strengths and weaknesses.

Landmarks (<https://matatk.agrip.org.uk/landmarks/>) is an extension available for Firefox, Chrome, Opera, and Edge that focuses on navigation via landmarks. This can be a great way to do an initial check of the landmarks on your page and see if they're improving navigation or making it worse.

I'm a big fan of an article called **WAI-ARIA and its true impact on assistive technologies** (<https://tech.finn.no/2017/06/07/wai-aria-and-its-true-impact-on-assistive-technologies/>) by Tor-Martin Storsletten. It's an older article but it contains an extensive, in-depth review of ARIA, how it's used, how it interacts with assistive technology like screen readers, and where it falls short.

Of course, I couldn't neglect the article where Sara Soueidan coined the phrase "ARIA is a polyfill for HTML semantics", **What a Year of Learning and Teaching Accessibility Taught Me** (<https://www.24a11y.com/2019/what-a-year-of-learning-and-teaching-accessibility-taught-me/>). The site it's published on, 24 Accessibility, has many great accessibility resources, including a very thorough and well-researched one on why there's really no replacing the standard select element (<https://www.24a11y.com/2019/select-your-poison/>).

Use ARIA in your applications, but use it wisely, with consideration and planning. Accessibility is a broad and tricky topic, and you're going to make mistakes. At the very least, try to make informed mistakes, and of course, do better once you know better.

Ashleigh Lodge
CODE

Can an LLM Make a Video Game?

In the Summer of 1980, I played Asteroids at a gas station in rural West Texas. I stood on a stool to reach the controls and see the screen. Ever since then, I've wanted to make a video game. I've also wanted to have the time, skills, and resources to make a video game. In recent years, it's become much easier to bring your idea to life using tools like Godot or one of the GameMaker



Jason Murphy

Thestrangerous.substack.com
@jason-murphy.bsky.social

Jason is a writer, AI enthusiast, and media creator. As the producer and co-host of **Hacking the System** on the National Geographic Channel, he stole cars in Hollywood, made improvised smoke bombs, and prepared for the apocalypse. On YouTube, he co-created and hosted **the Modern Rogue**, where he explored hacking, lock-picking, and trade-craft. After publishing multiple speculative fiction novels and writing a produced screenplay, Jason is currently exiled in the desert, where he stares into the future with awe and terror. And he still wants to make a video game.



engines, even with very little coding experience. That's all well and good, but in the interests of curiosity and laziness, I want to know if I can prompt a Large Language Model to do it all for me.

My focus is to see just what I can coax out of an LLM to replicate a vintage Asteroids experience. I'm fairly confident I can get assistance with boilerplate blocks of code for most of it. From there, cobbling together the final game won't be a problem, but with only prompting input on my part, how close can I get? Can I generate entire

game elements? Asteroids is a simple game. Can an LLM rebuild it all, tabula rasa?

For this experiment, I'll try to stick to just prompting and using whatever the LLM provides. Along the way, I'll take a look at the code. Invariably, inefficiencies are going to jump out at you. You'll see errors start to pop up as you move forward. You may see the LLM inexplicably change code that was working just fine. But you can't change it! That's against the rules. You have to get LLM to do it using plain language requests.



The Plan

Not all LLMs are made the same, of course. For this, I'll use ChatGPT 4o (the paid version), accessible via OpenAI's website or other integrated platforms. With any LLM, clarity and specificity in your prompts are critical to getting useful responses. The more precise your prompt, the better your output. In keeping within the boundaries of this experiment, I'll just ask ChatGPT how to proceed:

"I'd like to recreate the classic arcade game, Asteroids, in Python. I will ask for your help coding this, but can we first come up with a plan on how to best approach the experiment?"

As you can see in **Figure 1**, ChatGPT immediately comes back with a clear and concise approach. I had a pretty good idea of how to tackle the project and what questions to ask the LLM, but ChatGPT instantly offers a number of elements I hadn't considered.

Simple and elegant, I couldn't find any issues with its suggested plan of attack. First, ChatGPT calls the shots. It analyzes all of the elements of the classic game that I'll need to clone. There are three pillars for this project: the gameplay, the controls, and the game objects.

The LLM sums up the gameplay in a few concise sentences. The player pilots a ship that can rotate and thrust. The goal is to destroy the asteroids. When shot, they break into smaller pieces. End game criteria is met when the player's ship collides with an asteroid. It's recommended that you use arrow keys to control the ship and the space bar to shoot. Bog standard, as they say.

With a game this basic, it's easy to identify the various game objects. There's a ship that rotates, thrusts, and shoots on the player's command. The asteroids must move randomly and break up when they're hit, and the bullets used to shoot them need to be defined. If you want to get fancy, ChatGPT suggests adding additional enemies.

As it continues, the LLM lists the tools you'll need, the Python libraries of Pygame and Math. You're given a step-by-step project breakdown, followed by suggested development iterations and a basic reminder to debug regularly. One of my favorite things about these sorts of projects is shown in **Figure 2**. ChatGPT prompts you to jump right in.

The Code

After some experience with LLMs, I've learned that it's wise to monitor your language. It's like making a wish to a genie. The old parables say that the djinn will exploit any vagueness in your wording. Maybe you've encountered that in the form of a malicious DM during a D&D session. Fortunately, ChatGPT is much more forgiving. If I want to continue to specify parameters, I can. In fact, if I ask the LLM to ask me how it can do better, I might get much more accurate results.

Environment Set Up and Game Loop

Next, ChatGPT enthusiastically begins with the basics. Because I've asked for explanations, it holds my hand the entire way. "Great! Let's start by setting up the environment and creating a basic game loop in Python using the Pygame library. The game loop is the core of any

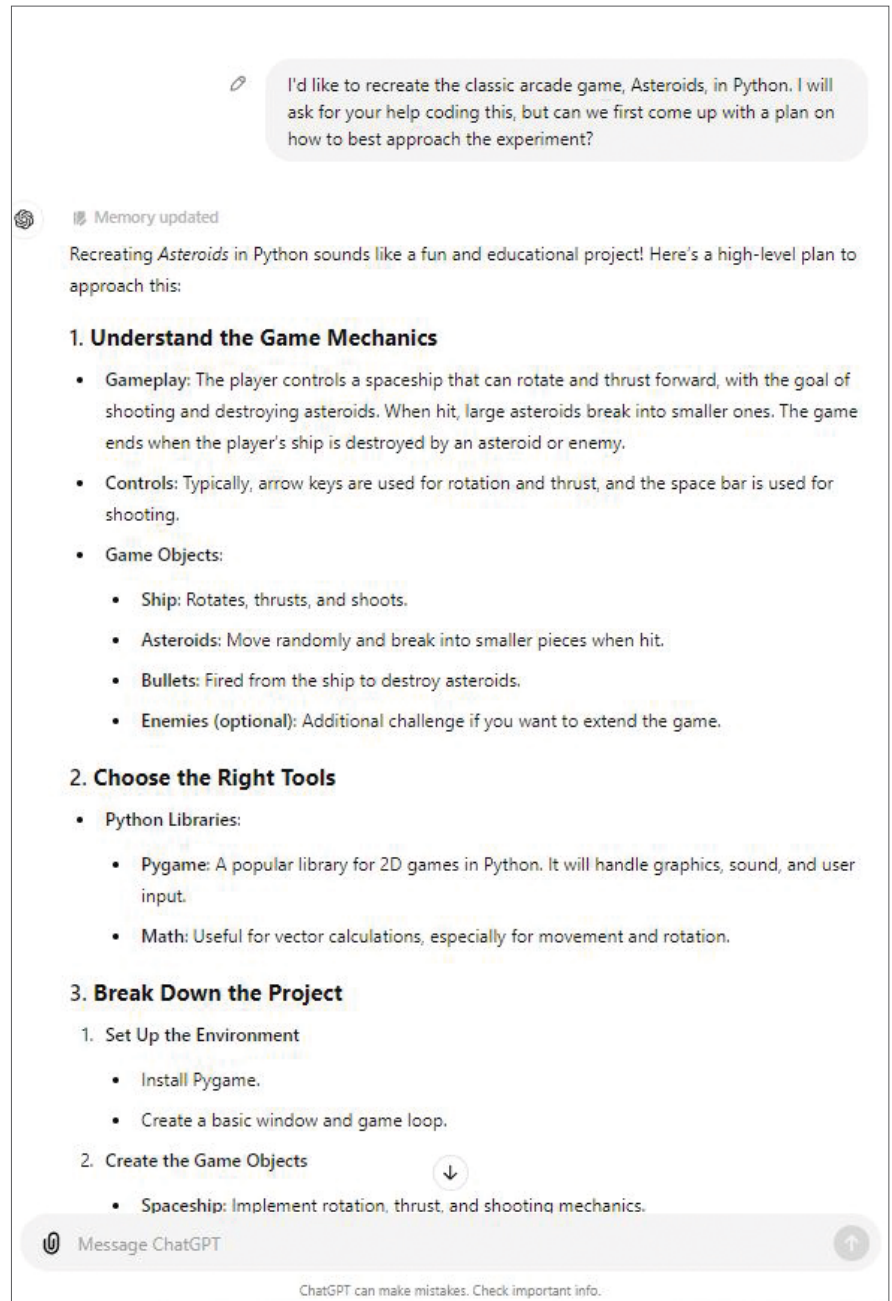


Figure 1: ChatGPT's plan

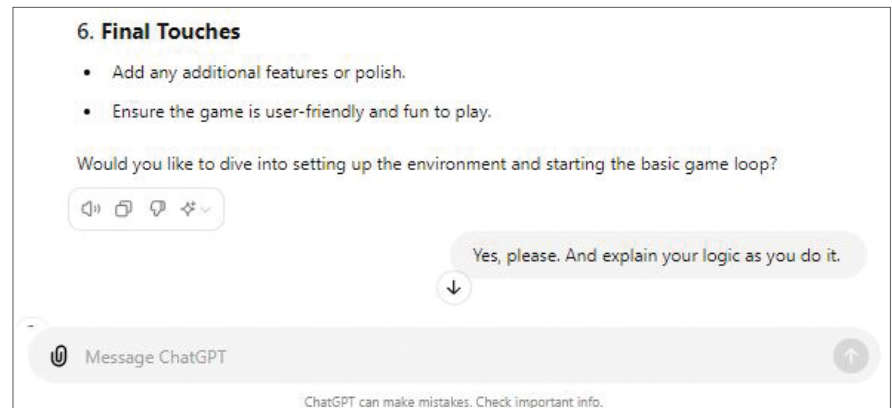


Figure 2: ChatGPT is ready to go.

Listing 1: Game Loop Listing

```
import pygame
import sys

# Initialize Pygame
pygame.init()

# Set up the game window
screen_width = 800
screen_height = 600
screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption("Asteroids")

# Set up the clock for controlling the frame rate
clock = pygame.time.Clock()
fps = 60

# Main game loop
running = True
while running:
    # Event handling

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Game logic (will be added later)

    # Clear the screen with a black color
    screen.fill((0, 0, 0))

    # Rendering (will be added later)

    # Update the display
    pygame.display.flip()

    # Cap the frame rate
    clock.tick(fps)

# Quit the game
pygame.quit()
sys.exit()
```

1. Set Up the Environment

Step 1: Install Pygame

First, you need to install Pygame. If you haven't installed it yet, you can do so using pip:

```
bash
pip install pygame
```

Step 2: Create the Basic Structure

Now, we'll create a simple Python script that initializes Pygame, sets up a game window, and runs a basic game loop.

Here's a breakdown of what we'll do:

1. **Initialize Pygame:** Start the Pygame modules.
2. **Create a Game Window:** Set the dimensions of the game window.

Figure 3: ChatGPT sets up Pygame.

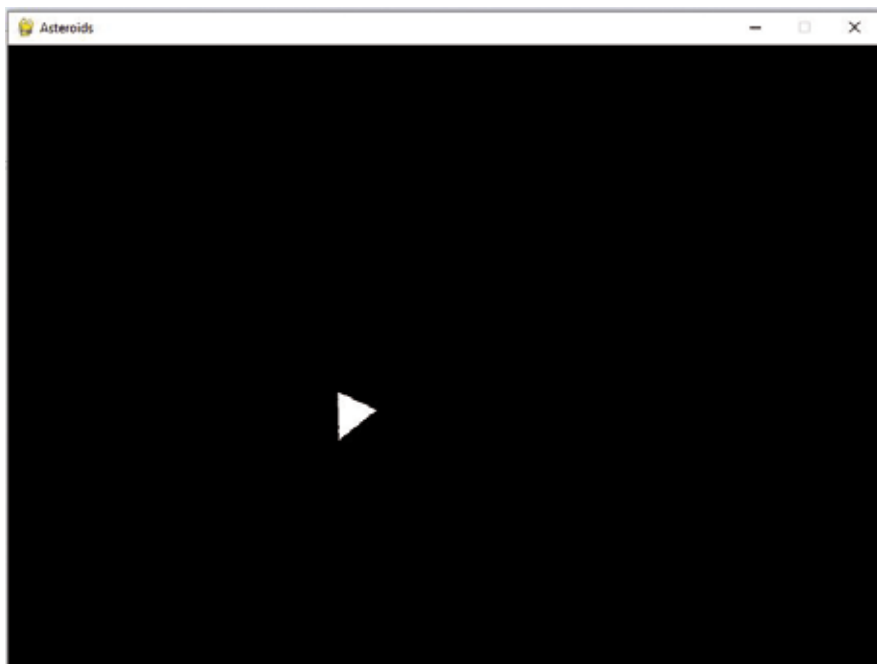


Figure 4: The ship in the vast void of space.

game, where the game continually updates and renders the screen.”

Install your Pygame and Math libraries, if you haven't already. ChatGPT starts there and provides a bulleted list of what its code will accomplish. It tells you that you're going to first address items like the main game loop, basic event handling, and rendering. Then it's off to the races! In **Figure 3**, you'll see the ChatGPT interface and how it presents basic copyable code.

See **Listing 1** for the full code that ChatGPT provided. It's as bare bones as you can imagine, but you have to appreciate the copious amounts of commenting. Next, it responds to my request to explain its logic. Here, the LLM shows potential to really shine as an instructional tool. It justifies every line of code with a basic explanation.

- The **while running** loop keeps the game running as long as **running** is True.
- It loops through all events using **pygame.event.get()**. If the event is **pygame.QUIT**, which occurs when the player closes the window, set **running** to False to exit the loop.

There's not much to get wrong here. It's basic, but nothing crashes when you run it. Time to move on. Per the plan provided, after the explanation of the code, ChatGPT offers to move on to the next step.

Creating the Spaceship and Controls

“Would you like to move on to creating the spaceship and implementing its controls next?”

“Yes!”

That was the entirety of my response, an enthusiastic “Yes!” ChatGPT has the wheel. Time to set up the first class and add a spaceship. As with the last section, ChatGPT says what it's going to code, provides that code, and then justifies it in detail. It begins by defining the Spaceship class, giving thought to things like speed, acceleration, size, and the like. These choices may seem arbitrary, but they're based on the part of ChatGPT's massive dataset that involves the classic video game. The class for the ship provides **almost** everything

ChatGPT suggested. In the initial plan, the LLM identified the core features of the spaceship. The ship needs to be able to rotate, thrust, and shoot. If you look at the code or the explanation provided, you'll note that in this iteration, you don't yet have the ability to shoot the titular asteroids. At this point, I don't know if ChatGPT has forgotten that aspect, or if it'll implement the bullets later. After the class is created, ChatGPT creates the instance of the spaceship and adds the ship to the main game loop. In the downloadable file, **Spaceship and Controls**, you can see this iteration of the single game file. If I run the code, it works as advertised. Check out **Figure 4**.

The ship is an isosceles triangle. That's appropriate, but rather than use the vertex angle as the nose of the ship, it uses one of the sides of the triangle. The ship only flies sideways! I could wrestle with this and try to get ChatGPT to fix it, but it's playable. I'll live with it for this iteration.

After its explanation of the code, ChatGPT offers a choice: "Would you like to add boundaries so the spaceship wraps around the screen when it goes off the edges, or would you prefer to work on shooting mechanics next?"

I'm sorry I doubted you, ChatGPT.

Shooting Mechanics

I choose bullets! For the bullets, ChatGPT creates a new **Bullet** class and modifies the **Spaceship** class to shoot those bullets via a press of the space bar. In the downloadable file, **Shooting Mechanics**, you can review the provided code. It works. I can't say that it works well, but it works. Take a look at **Figure 5** to see what I mean.

Notice anything? The bullets are coming from the side of the ship. Because ChatGPT set the side of the triangle as the front of the ship, that makes sense and I suppose I can make that work, but it's certainly disorienting. This is the kind of strange flaw you'll encounter when trying a project like this. It technically works, but it's still in the uncanny valley of code.

With the next step, ChatGPT suggests you work on what happens when you shoot an asteroid. It's flexible enough for you to jump around, however, and offers the chance to do so.

"Would you like to add collision detection between bullets and asteroids next, or is there another feature you'd like to work on?"

I stick to the path. "I would like to add boundaries, so the spaceship wraps around the screen when it goes off the edges."

Game Boundaries

From here, ChatGPT updates the code for the **Spaceship** class, enabling the ship to appear on one side of the screen after it flies off the other. Once again, each addition made to the code is accompanied by a helpful note. This phase, labeled **Boundary Wrapping**, is available on the CODE Magazine website with this article. In the explanation of logic, it's explained how to update the ship's position once it has gone offscreen.

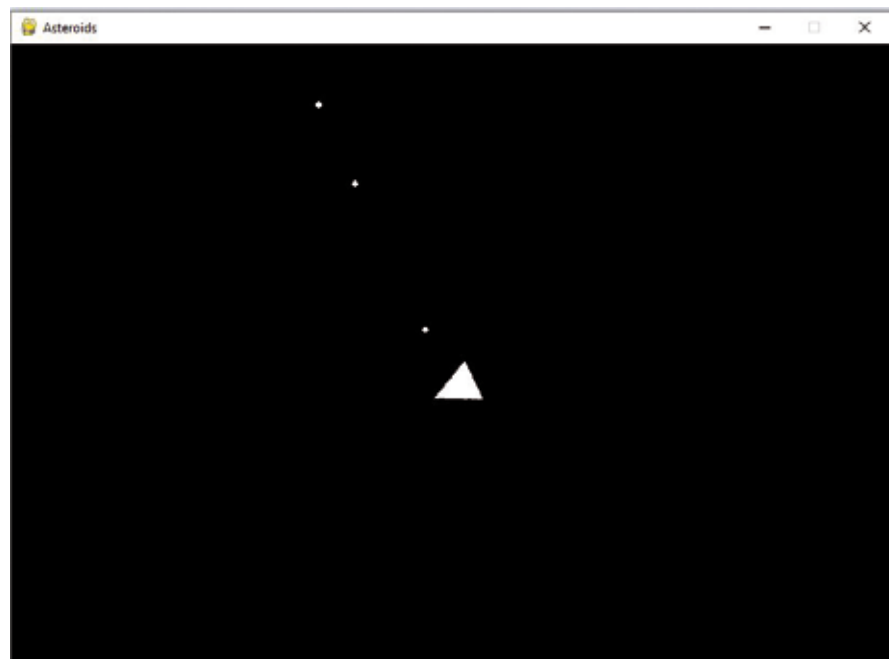


Figure 5: Now there are bullets.

- If **self.x < 0**, the spaceship has gone off the left edge, so wrap it to the right edge by setting **self.x = screen_width**.
- If **self.x > screen_width**, the spaceship has gone off the right edge, so wrap it to the left edge by setting **self.x = 0**.
- Similarly, if **self.y < 0**, the spaceship wraps from the top to the bottom when you set **self.y = screen_height**.
- If **self.y > screen_height**, the spaceship wraps from the bottom to the top by setting **self.y = 0**.

After each iteration, I run the code to test. So far so good. Nothing flashy, but no errors, either.

"Let's add asteroids," I prompt.

The Asteroids

As instructed, ChatGPT remembers to explain what's needed to do in this round of code. It defines what's required from the **Asteroid** class, providing a list of behaviors and features to account for.

```
class Asteroid:
    def __init__(self, x, y, size):
        self.x = x
        self.y = y
        self.size = size
        self.speed = random.uniform(1, 3)
        self.angle = random.uniform(0, 360)
        self.color = (200, 200, 200)
```

To run this version, grab the downloadable code named **Adding the Asteroids**. When you run it, you get what you see in **Figure 6**. The asteroids are spheres. The ship is a triangle. But it works. With each iteration, I feel like I'm building a house of cards. What will be too much for this language model to handle? Will the collision detection I need to add send this all tumbling down?

A Link to the Conversation

I've provided the link to the unedited discussion with ChatGPT. It merits mentioning that the responses given may have been colored by what it knows about me. As someone who uses it regularly, ChatGPT has a lot of details about my projects, experience, and interests. The responses you receive may be similarly influenced.

Even subtle differences in your wording can produce different results. It's important that you are specific with how you want to approach any project.

<https://chatgpt.com/share/54a13835-043b-4d4f-83b9-7bdb86a1b954>

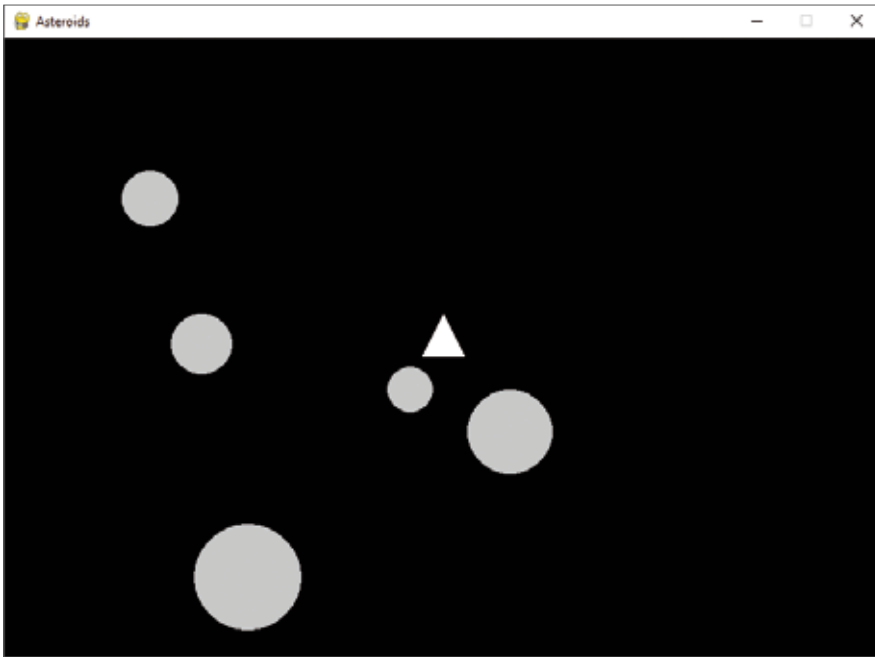


Figure 6: You're not actually going INTO an asteroid field!?

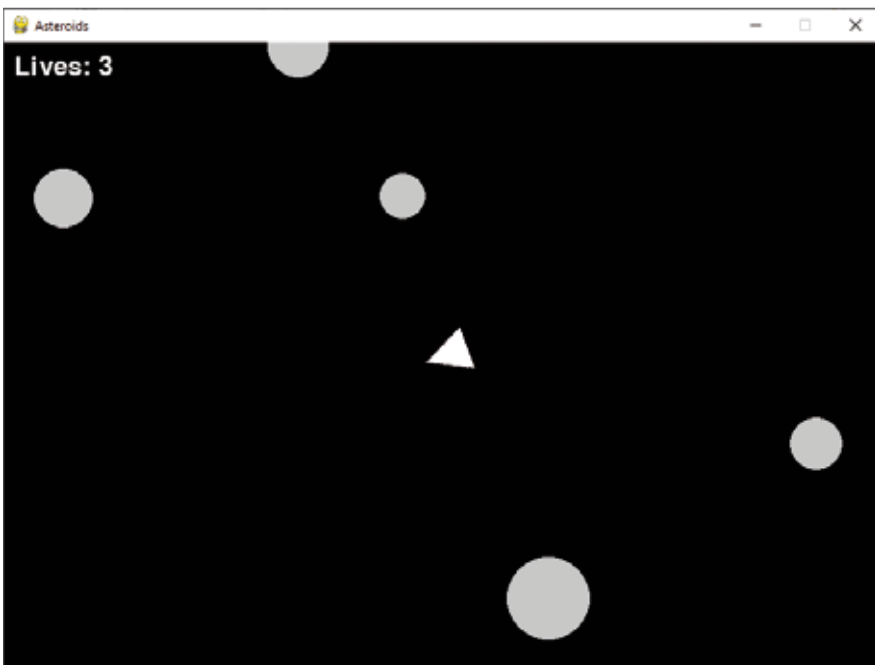


Figure 7: Player lives were added in the upper left.

One thing jumped out at me about this last iteration.

```
import pygame
import sys
import math
import random
```

This is the first time it's called the Python library **random** to generate random numbers. Because ChatGPT has been so good at explaining all of its choices, I double check the **Explanation of Logic** it provides after every iteration. There's nothing in there detailing why it did this.

Of course, you and I know it's for the various qualities of the asteroids themselves—speed, size, position—but because I haven't been inspecting every line of changed code, I have to wonder what other decisions it's made without telling me.

Collision Detection

"Would you like to add collision detection next so that bullets can destroy asteroids, or is there another feature you want to work on?"

"Yes. Collision detection, please."

This iteration of the project is found in the downloadable Python file named **Collision Detection**. Here, ChatGPT gives you the ability to blow stuff up. Now you have a game!

Note that within the adjusted **Bullet** class, ChatGPT provides this update:

```
def collides_with(self, asteroid):
    distance = math.hypot(self.x -
        asteroid.x, self.y - asteroid.y)
    return distance < self.radius +
        asteroid.size
```

The big spheres now break up into little spheres when they collide with the bullets, a key element in the classic game.

Player Lives

Here, ChatGPT guides you to select the next aspect of the game to add. You can interrupt if you see something going wrong or heading in an unhelpful direction, but so far, it all makes sense.

"Would you like to add any more features, such as a score system, player lives, or something else?"

Player lives seem a bit more important than the score, so I'm going to start there, move on to scoring, and then reassess to see what else I might need.

A few things need to happen to implement player lives. ChatGPT breaks it down:

1. Modify the **Spaceship** class to Track Lives.
2. Implement Collision Detection between the Spaceship and the Asteroids.
3. Display the Number of Lives on screen.

The code ChatGPT provided can be seen in the downloadable file **Adding Player Lives** and the on-screen result in **Figure 7**.

The logic used is extraordinarily straightforward. ChatGPT explains that I added a **Lives** attribute to the **Spaceship** class. I added a method to check for collisions with asteroids.

If a collision is detected?

```
('spaceship.lives -= 1')
```

If the player runs out of lives?

```
('running = False')
```

Keeping Score

After we add the player lives, ChatGPT keeps moving forward, suggesting adding a scoring feature next.

"A scoring system would be great," is the only thing I need to tell it. It proceeds to walk us through adding the **Score** attribute to the **Spaceship** class and a function to increase the score based on the size of the asteroid.

There are a few key bits of code to point out in this portion. Please note that some of the code snippets presented throughout may have been broken to accommodate the limitations of print. If you'd like to copy and paste, please put the code into an unbroken line.

```
spaceship.score += asteroid.size based on asteroid size
```

and

```
# Display the score
score_text =
    font.render(f"Score: {spaceship.score}",
True, (255, 255, 255))
screen.blit
(score_text, (screen_width - 150, 10))
```

The code for this section is seen in the **Keeping Score** download. As you can see in **Figure 8**, it runs flawlessly.

Game Over

In minutes, you've built enough of a game to need a **Game Over** screen. The updated code it provides is in **Listing 2**. In **Figure 9**, you can see what happens when you try to run the game after its latest iteration.

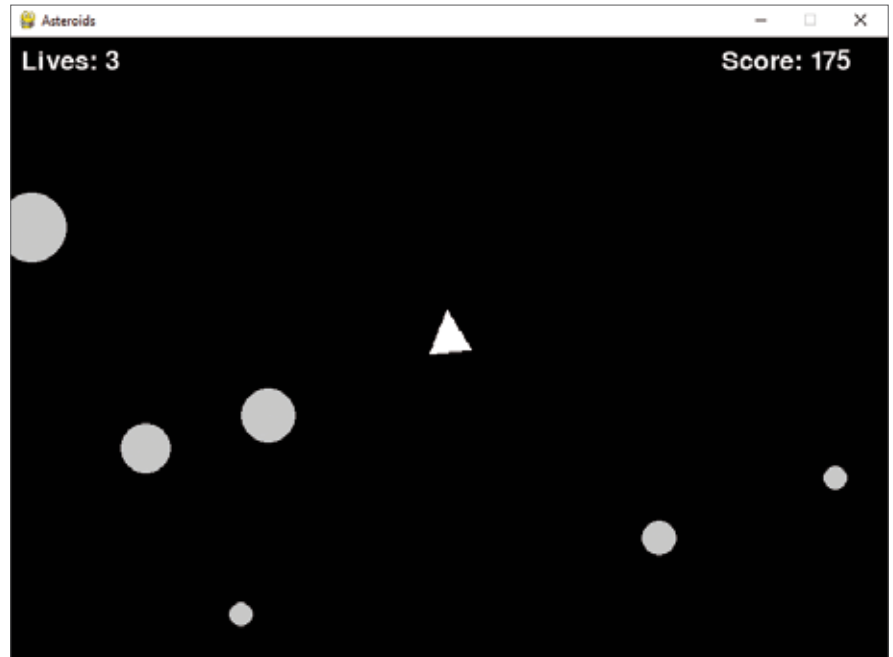


Figure 8: The Score is in the upper right.

```
Traceback (most recent call last):
  File "D:\VS Code Stuff\python\asteroids\asteroidsILM.py", line 13, in <module>
    pygame.display.setCaption("Asteroids")
    ~~~~~
AttributeError: module 'pygame.display' has no attribute 'setCaption'. Did you mean: 'set_caption'?
PS D:\VS Code Stuff\python\asteroids>
```

Figure 9: The first error

Listing 2: Game Over Screen Listing

```
import pygame
import sys
import math
import random

# Initialize Pygame
pygame.init()

# Set up the game window
screen_width = 800
screen_height = 600
screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.setCaption("Asteroids")

# Set up the clock for controlling the frame rate
clock = pygame.time.Clock()
fps = 60

# Set up the font for displaying lives, score, and game over text
font = pygame.font.SysFont(None, 36)
game_over_font = pygame.font.SysFont(None, 72)

# Define the Bullet class
class Bullet:
    def __init__(self, x, y, angle):
        self.x = x
        self.y = y
        self.angle = angle
        self.speed = 10
        self.radius = 3
        self.color = (255, 255, 255)

    def move(self):
        radians = math.radians(self.angle)
        self.x += self.speed * math.cos(radians)
        self.y -= self.speed * math.sin(radians)

        self.y -= self.speed * math.sin(radians)

    def draw(self):
        pygame.draw.circle(screen, self.color,
(int(self.x), int(self.y)), self.radius)

    def is_off_screen(self):
        return self.x < 0 or self.x > screen_width or
self.y < 0 or self.y > screen_height

    def collides_with(self, asteroid):
        distance = math.hypot(self.x - asteroid.x,
self.y - asteroid.y)
        return distance < self.radius + asteroid.size

# Define the Asteroid class
class Asteroid:
    def __init__(self, x, y, size):
        self.x = x
        self.y = y
        self.size = size
        self.speed = random.uniform(1, 3)
        self.angle = random.uniform(0, 360)
        self.color = (200, 200, 200)

    def move(self):
        radians = math.radians(self.angle)
        self.x += self.speed * math.cos(radians)
        self.y -= self.speed * math.sin(radians)

        # Wrap around the screen
        if self.x < 0:
            self.x = screen_width
        elif self.x > screen_width:
            self.x = 0
```

Listing 2: continued

```

    if self.y < 0:
        self.y = screen_height
    elif self.y > screen_height:
        self.y = 0

    def draw(self):
        pygame.draw.circle(screen, self.color,
            (int(self.x), int(self.y)), self.size)

    def split(self):
        if self.size > 20: # Minimum size to split
            size = self.size // 2
            return [Asteroid(self.x, self.y, size),
                Asteroid(self.x, self.y, size)]
        else:
            return [] # Return an empty list if the
asteroid is too small to split

# Define the Spaceship class
class Spaceship:
    def __init__(self):
        self.x = screen_width // 2
        self.y = screen_height // 2
        self.angle = 0
        self.speed = 0
        self.rotation_speed = 5
        self.acceleration = 0.1
        self.max_speed = 5
        self.lives = 3 # Start with 3 lives
        self.score = 0 # Start with a score of 0
        self.image = pygame.Surface((40, 40))
        pygame.draw.polygon(self.image, (255, 255, 255),
            [(20, 0), (0, 40), (40, 40)])
        self.image.set_colorkey((0, 0, 0))
        self.bullets = []

    def rotate(self, direction):
        if direction == "left":
            self.angle += self.rotation_speed
        elif direction == "right":
            self.angle -= self.rotation_speed

    def thrust(self):
        radians = math.radians(self.angle)
        self.x += self.speed * math.cos(radians)
        self.y -= self.speed * math.sin(radians)
        self.speed = min(self.speed +
            self.acceleration, self.max_speed)

        # Wrap around the screen
        if self.x < 0:
            self.x = screen_width
        elif self.x > screen_width:
            self.x = 0
        if self.y < 0:
            self.y = screen_height
        elif self.y > screen_height:
            self.y = 0

    def shoot(self):
        bullet = Bullet(self.x, self.y, self.angle)
        self.bullets.append(bullet)

    def draw(self):
        rotated_image =
pygame.transform.rotate(self.image, self.angle)
        rect = rotated_image.get_rect(center=(self.x, self.y))
        screen.blit(rotated_image, rect.topleft)

    def update(self, keys):
        if keys[pygame.K_LEFT]:
            self.rotate("left")
        if keys[pygame.K_RIGHT]:
            self.rotate("right")
        if keys[pygame.K_UP]:
            self.thrust()

    def reset_position(self):
        self.x = screen_width // 2
        self.y = screen_height // 2
        self.speed = 0

        self.angle = 0

    def collides_with(self, asteroid):
        distance = math.hypot(self.x - asteroid.x,
            self.y - asteroid.y)
        return distance < 20 + asteroid.size # 20 is the approximate size of
the spaceship

    def game_over_screen(spaceship):
        screen.fill((0, 0, 0))

        game_over_text = game_over_font.render("GAME OVER", True,
            (255, 255, 255))
        score_text = font.render(f"Final Score:
{spaceship.score}", True, (255, 255, 255))
        restart_text = font.render("Press R to Restart
or Q to Quit", True, (255, 255, 255))

        screen.blit(game_over_text, (screen_width //
            2 - game_over_text.get_width() //
            2, screen_height // 3))
        screen.blit(score_text, (screen_width //
            2 - score_text.get_width() //
            2, screen_height // 2))
        screen.blit(restart_text, (screen_width //
            2 - restart_text.get_width() //
            2, screen_height // 2 + 50))

        pygame.display.flip()

        while True:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_r:
                        return True # Restart the game
                    elif event.key == pygame.K_q:
                        pygame.quit()
                        sys.exit()

    def reset_game():
        global spaceship, asteroids
        spaceship = Spaceship()
        asteroids = []
        for _ in range(5): # Start with 5 asteroids
            x = random.randint(0, screen_width)
            y = random.randint(0, screen_height)
            size = random.randint(20, 50)
            asteroid = Asteroid(x, y, size)
            asteroids.append(asteroid)

        # Create an instance of the spaceship
        spaceship = Spaceship()

        # Create a list of asteroids
        asteroids = []
        for _ in range(5): # Start with 5 asteroids
            x = random.randint(0, screen_width)
            y = random.randint(0, screen_height)
            size = random.randint(20, 50)
            # Asteroids will have a size between 20 and 50
            asteroid = Asteroid(x, y, size)
            asteroids.append(asteroid)

        # Main game loop
        running = True
        while running:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_SPACE:
                        spaceship.shoot()

            keys = pygame.key.get_pressed()
            spaceship.update(keys)

            for bullet in spaceship.bullets[:]:
                bullet.move()

```

Listing 2: continued

```
if bullet.is_off_screen():
    spaceship.bullets.remove(bullet)
else:
    for asteroid in asteroids[:]:
        if bullet.collides_with(asteroid):
            spaceship.bullets.remove(bullet)
            new_asteroids = asteroid.split()
            spaceship.score += asteroid.size
            asteroids.remove(asteroid)
            asteroids.extend(new_asteroids)
            break

for asteroid in asteroids:
    asteroid.move()
    if spaceship.collides_with(asteroid):
        spaceship.lives -= 1
        if spaceship.lives <= 0:
            if game_over_screen(spaceship):
                reset_game()
            else:
                running = False
        else:
            spaceship.reset_position()

screen.fill((0, 0, 0))
spaceship.draw()

for bullet in spaceship.bullets:
    bullet.draw()

for asteroid in asteroids:
    asteroid.draw()

lives_text = font.render(f"Lives: {spaceship.lives}",
True, (255, 255, 255))
screen.blit(lives_text, (10, 10))

score_text = font.render(f"Score: {spaceship.score}",
True, (255, 255, 255))
screen.blit(score_text, (screen_width - 150, 10))

pygame.display.flip()
clock.tick(fps)

pygame.quit()
sys.exit()
```

You have your first failure to launch. Visual Studio immediately shows you the problem, which is an easy fix. You may ask, “Why that error?” As the project grows, so do the chances for errors, of course, but that code was settled. It was working. You didn’t need to change a thing! Here’s where you start to see the cracks form. Rather than go in and fix it, let’s stay true to the mission and ask ChatGPT to do it.

Errors and Debugging

Here’s what I provide to ChatGPT:

“We have an error.

```
AttributeError: module 'pygame.display'
has no attribute 'setCaption'.
Did you mean: 'set_caption'?"
```

Rather than regurgitate the entire file, ChatGPT sees what it did wrong and provides the exact fix, as seen in **Figure 10**.

After that easy fix, the game launches without issue. With lives, a score, and a **Game Over** screen, it’s starting to feel like a real game, as you can see in **Figure 11**. Sure, it’s nothing pretty, and the Asteroids code is dirt simple, but the original file size was only 8KB.

After this, you get a mostly functional game, but it’s a little wonky. I notice that sometimes the game ends after the first death. I ask ChatGPT about it.

“Now I get to the game over screen after the first death, rather than after three lives. How can I fix that?”

It immediately suggests a revised code block, as seen below:

```
for asteroid in asteroids:
    asteroid.move()
    if spaceship.collides_with(asteroid):
        spaceship.lives -= 1
        if spaceship.lives > 0:
```

It looks like there's a typo in the code. The correct method is `pygame.display.set_caption()` instead of `pygame.display.setCaption()`.

Here's the corrected line:

```
python
pygame.display.set_caption("Asteroids")
```

Replace the line with the correct method name, and it should resolve the error.

Let me know if you encounter any other issues or if there's something else you'd like to work on!

🔊 📄 ↻ 🗑️ ✨

Figure 10: Debugging with ChatGPT



Figure 11: Game Over.

SPONSORED SIDEBAR

AI Executive Briefing

Experience the game-changing impact of **AI** through **CODE Consulting's** Executive Briefing service. Uncover the immense potential and wide-ranging benefits of **AI** in every industry. Our briefing provides **strategic guidance** for seamless implementation, covering crucial aspects such as infrastructure, talent acquisition, and leadership.

Discover how to effectively **integrate AI** and propel your organization into future success. **Contact us today** to schedule your executive briefing and embark on a journey of AI-powered growth. www.codemag.com/ai

```
spaceship.reset_position() # Reset
the spaceship position if there are still lives
left
    else:
        if game_over_screen(spaceship):
# Only call the game over screen when lives are
<= 0
            reset_game() # Restart the game
        else:
            running = False # End the game
if the player chooses to quit
```

After this, I notice something. I respond to ChatGPT in plain language, "Now the asteroids don't seem to collide with the ship."

ChatGPT comes back with why it thinks the error is occurring and immediately offers a fix. Here's the entire exchange:

If the asteroids are not colliding with the ship as expected, the issue might be related to the collision detection logic between the ship and the asteroids. Let's review and refine the collision detection method.

Collision Detection Review

The `collides_with` method in the **Spaceship** class checks if the distance between the spaceship and an asteroid is less than the sum of their radii. If the collision detection isn't working correctly, it could be due to the size of the spaceship being incorrectly accounted for.

Steps to Fix the Collision Detection

- 4. Ensure Proper Size Calculation:** Make sure the size of the spaceship is correctly factored into the collision detection. The collision should occur when the distance between the center of the spaceship and the asteroid is less than the sum of half the spaceship's size and the asteroid's size.
- 5. Adjust the Collision Detection Method:** Let's update the `collides_with` method in the **Spaceship** class to use the correct size and check for collisions more accurately.

Updated collides_with Method

Here's an updated version of the `collides_with` method for the **Spaceship** class:

```
def collides_with(self, asteroid):
    # Calculate the distance between the spaceship
    and the asteroid
    distance = math.hypot(self.x -
asteroid.x, self.y - asteroid.y)
    # Consider the spaceship size as half of its
    width
    spaceship_size = 20 # Approximate size of
the spaceship
    # Check if the distance is less than the sum
of their radii
    return distance < spaceship_size
+ asteroid.size
```

Explanation of Changes:

- Spaceship Size:** The spaceship size is set to 20, representing half the width of the triangle-shaped spaceship. You can adjust this value if the spaceship's actual size differs.
- Collision Logic:** The collision occurs if the distance between the spaceship and the asteroid is less than the sum of the spaceship's radius (approximately half its size) and the asteroid's radius (asteroid.size).

Here's how this updated collision detection method integrates into the game loop:

```
for asteroid in asteroids:
    asteroid.move()
    if spaceship.collides_with(asteroid):
        spaceship.lives -= 1
        if spaceship.lives > 0:
            spaceship.reset_position() # Reset
the spaceship position if there are still
lives left
        else:
            if game_over_screen(spaceship):
# Only call the game over screen when lives
are <= 0
                reset_game() # Restart the game
            else:
                running = False # End the game if
the player chooses to quit
```

"After implementing this fix, run the game again. The asteroids should now properly collide with the spaceship, reducing lives and eventually leading to the Game Over screen when all lives are lost. Let me know if this resolves the issue or if there's anything else you'd like to adjust!"

Unfortunately, the problems don't stop there. Now the game goes straight to the Game Over screen. Each fix breaks something else. You know the drill. Each time I come back to it with an error, ChatGPT provides not only why it might not be working, but how to fix it. Throughout the process, the LLM even starts to implement print statements to help debug.

```
def collides_with(self, asteroid):
    distance = math.hypot(self.x - asteroid.x,
self.y - asteroid.y)
    spaceship_size = 20 # Approximate size
of the spaceship
    is_collision = distance <
spaceship_size + asteroid.size

    # Debug print to check collision detection
    print(f"Checking collision: Distance =
{distance}, Threshold = {spaceship_size + asteroid.size},
Collision = {is_collision}")

    return is_collision
```

As an insecure coder, I often start to believe that I'm the problem. This is one of those times. I was getting into the weeds. I take a step back and request, "Please rebuild the complete file with all of these changes."

It responds immediately with the code seen in **Listing 3**.

Listing 3: Full Game Listing

```
import pygame
import sys
import math
import random

# Initialize Pygame
pygame.init()

# Set up the game window
screen_width = 800
screen_height = 600
screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption("Asteroids")

# Set up the clock for controlling the frame rate
clock = pygame.time.Clock()
fps = 60

# Set up the font for displaying lives, score, and game over text
font = pygame.font.SysFont(None, 36)
game_over_font = pygame.font.SysFont(None, 72)

# Define the Bullet class
class Bullet:
    def __init__(self, x, y, angle):
        self.x = x
        self.y = y
        self.angle = angle
        self.speed = 10
        self.radius = 3
        self.color = (255, 255, 255)

    def move(self):
        radians = math.radians(self.angle)
        self.x += self.speed * math.cos(radians)
        self.y -= self.speed * math.sin(radians)

    def draw(self):
        pygame.draw.circle(screen, self.color,
                           (int(self.x), int(self.y)), self.radius)

    def is_off_screen(self):
        return self.x < 0 or self.x > screen_width or
               self.y < 0 or self.y > screen_height

    def collides_with(self, asteroid):
        distance = math.hypot(self.x - asteroid.x,
                              self.y - asteroid.y)
        return distance < self.radius + asteroid.size

# Define the Asteroid class
class Asteroid:
    def __init__(self, x, y, size):
        self.x = x
        self.y = y
        self.size = size
        self.speed = random.uniform(1, 3)
        self.angle = random.uniform(0, 360)
        self.color = (200, 200, 200)

    def move(self):
        radians = math.radians(self.angle)
        self.x += self.speed * math.cos(radians)
        self.y -= self.speed * math.sin(radians)

        # Wrap around the screen
        if self.x < 0:
            self.x = screen_width
        elif self.x > screen_width:
            self.x = 0

    def draw(self):
        if self.y < 0:
            self.y = screen_height
        elif self.y > screen_height:
            self.y = 0

    def split(self):
        if self.size > 20: # Minimum size to split
            size = self.size // 2
            return [Asteroid(self.x, self.y, size),
                    Asteroid(self.x, self.y, size)]
        else:
            return [] # Return an empty list if
                       # the asteroid is too small to split

# Define the Spaceship class
class Spaceship:
    def __init__(self):
        self.x = screen_width // 2
        self.y = screen_height // 2
        self.angle = 0
        self.speed = 0
        self.rotation_speed = 5
        self.acceleration = 0.1
        self.max_speed = 5
        self.lives = 3 # Start with 3 lives
        self.score = 0 # Start with a score of 0
        self.image = pygame.Surface((40, 40))
        pygame.draw.polygon(self.image,
                            (255, 255, 255), [(20, 0), (0, 40), (40, 40)])
        self.image.set_colorkey((0, 0, 0))
        self.bullets = []

    def rotate(self, direction):
        if direction == "left":
            self.angle += self.rotation_speed
        elif direction == "right":
            self.angle -= self.rotation_speed

    def thrust(self):
        radians = math.radians(self.angle)
        self.x += self.speed * math.cos(radians)
        self.y -= self.speed * math.sin(radians)
        self.speed = min(self.speed +
                          self.acceleration, self.max_speed)

        # Wrap around the screen
        if self.x < 0:
            self.x = screen_width
        elif self.x > screen_width:
            self.x = 0
        if self.y < 0:
            self.y = screen_height
        elif self.y > screen_height:
            self.y = 0

    def shoot(self):
        bullet = Bullet(self.x, self.y, self.angle)
        print(f"Shooting bullet from {self.x}, {self.y}")
        # Debug bullet creation
        self.bullets.append(bullet)

    def draw(self):
        rotated_image =
pygame.transform.rotate(self.image, self.angle)
        rect = rotated_image.get_rect(center=(self.x, self.y))
        screen.blit(rotated_image, rect.topleft)
```

Listing 3: continued

```

def update(self, keys):
    if keys[pygame.K_LEFT]:
        self.rotate("left")
    if keys[pygame.K_RIGHT]:
        self.rotate("right")
    if keys[pygame.K_UP]:
        self.thrust()

def reset_position(self):
    self.x = screen_width // 2
    self.y = screen_height // 2
    self.speed = 0
    self.angle = 0

def collides_with(self, asteroid):
    distance = math.hypot(self.x - asteroid.x,
self.y - asteroid.y)
    spaceship_size = 20 # Approximate size
of the spaceship
    is_collision = distance < spaceship_size + asteroid.size
    print(f"Checking collision: Distance = {distance},
Threshold = {spaceship_size + asteroid.size}, Collision = {is_collision}")
    return is_collision

def game_over_screen(spaceship):
    screen.fill((0, 0, 0))

    game_over_text = game_over_font.render("GAME OVER", True
, (255, 255, 255))
    score_text = font.render(f"Final Score:
{spaceship.score}", True, (255, 255, 255))
    restart_text = font.render("Press R to Restart
or Q to Quit", True, (255, 255, 255))

    screen.blit(game_over_text, (screen_width
// 2 - game_over_text.get_width() // 2,
screen_height // 3))
    screen.blit(score_text,
(screen_width // 2 - score_text.get_width()
// 2, screen_height // 2))
    screen.blit(restart_text,
(screen_width // 2 - restart_text.get_width() // 2,
screen_height // 2 + 50))

    pygame.display.flip()

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_r:
                    return True # Restart the game
                elif event.key == pygame.K_q:
                    pygame.quit()
                    sys.exit()

def reset_game():
    global spaceship, asteroids
    spaceship = Spaceship()
    asteroids = []
    for _ in range(5): # Start with 5 asteroids
        while True:
            x = random.randint(0, screen_width)
            y = random.randint(0, screen_height)
            size = random.randint(20, 50)
            if math.hypot(x - spaceship.x, y - spaceship.y) >
100: # Ensure they don't spawn too close

            asteroid = Asteroid(x, y, size)
            asteroids.append(asteroid)
            break

# Create an instance of the spaceship
spaceship = Spaceship()

# Create a list of asteroids
asteroids = []
for _ in range(5): # Start with 5 asteroids
    while True:
        x = random.randint(0, screen_width)
        y = random.randint(0, screen_height)
        size = random.randint(20, 50) # Asteroids will
have a size between 20 and 50
        if math.hypot(x - spaceship.x, y - spaceship.y) >
100: # Ensure they don't spawn too close
            asteroid = Asteroid(x, y, size)
            asteroids.append(asteroid)
            break

# Main game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                spaceship.shoot()

    keys = pygame.key.get_pressed()
    spaceship.update(keys)

    for bullet in spaceship.bullets[:]:
        bullet.move()
        print(f"Bullet position: {bullet.x}, {bullet.y}")
    # Debug bullet position
    if bullet.is_off_screen():
        print("Removing bullet (off-screen)")
    # Debug bullet removal
    spaceship.bullets.remove(bullet)
    else:
        for asteroid in asteroids[:]:
            if bullet.collides_with(asteroid):
                print("Bullet hit asteroid!")
    # Debug bullet-asteroid collision
    spaceship.bullets.remove(bullet)
    new_asteroids = asteroid.split()
    spaceship.score += asteroid.size
    # Increase score based on asteroid size
    asteroids.remove(asteroid)
    asteroids.extend(new_asteroids)
    break

    for asteroid in asteroids:
        asteroid.move()
        if spaceship.collides_with(asteroid):
            spaceship.lives -= 1
            print(f"Lives remaining: {spaceship.lives}")
            if spaceship.lives > 0:
                spaceship.reset_position()
            else:
                print("Triggering Game Over Screen")
                if game_over_screen(spaceship):
                    reset_game()
                else:
                    running = False

    screen.fill((0, 0, 0))

```

Listing 3: continued

```
spaceship.draw()

for bullet in spaceship.bullets:
    bullet.draw()

for asteroid in asteroids:
    asteroid.draw()

lives_text = font.render(f"Lives: {spaceship.lives}",
True, (255, 255, 255))
screen.blit(lives_text, (10, 10))

score_text = font.render(f"Score: {spaceship.score}",
True, (255, 255, 255))
screen.blit(score_text, (screen_width - 150, 10))

pygame.display.flip()
clock.tick(fps)

pygame.quit()
sys.exit()
```

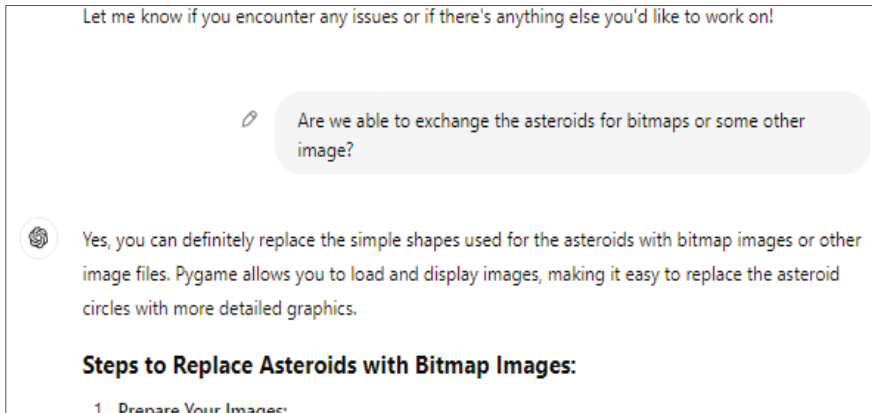


Figure 12: Enhancements

The Game

It works and it's actually kind of fun! Is this the best way to build an Asteroids clone? Absolutely not! But it's certainly a good litmus test to see how LLMs can be used going forward. As with all AI, you have to remember this mantra: This is as bad as it will ever get. The technology is advancing at a rate that's hard to conceive. As I was working on this article, Google released an AI generated version of Doom. It doesn't take a lot of imagination to see where this is heading.

There's one more thing I want to try. How easy would it be to replace those asteroids with images?

"Are we able to exchange the asteroids for bitmaps or some other image?" I ask in **Figure 12**.

This really kicks the game up to the next level. I can do away with the crude shapes and import my own pictures. I chose CODE Magazine's own Editor-In-Chief, Mr. Rod Pad-dock. He makes a great celestial hazard in **Figure 13**, don't you think?

In the end, with little human interference, I get a fun and recognizable rebuild of Asteroids. It's certainly not the best way to approach the task, but if used correctly, LLMs are a powerful coding partner. The link to the unedited chat with ChatGPT is provided in the sidebar. Although its limitations are on full display in this experiment, it also promises that a truly staggering change is coming, not just to game design, but coding in general.

Jason Murphy
CODE

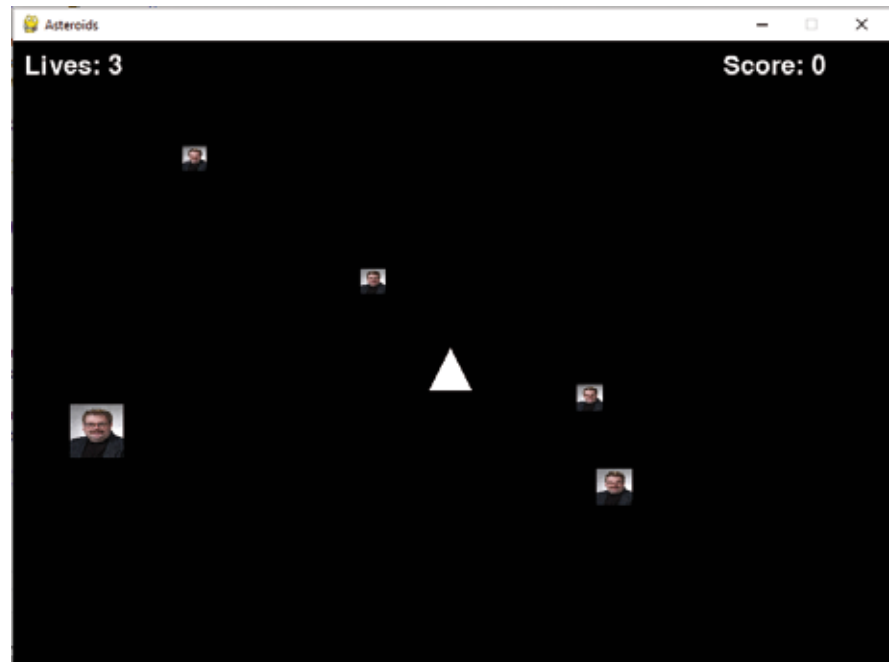


Figure 13: Importing images

Threads, Asynchrony, Parallelism, and Concurrency in C#

The concepts of process, thread, and task are fundamental to understanding the working of an operating system. You should have a good understanding of threads and how they work to learn asynchrony, parallelism, and concurrency. This article discusses the concepts related to these concepts in detail with relevant code examples wherever appropriate.



Joydip Kanjilal

joydipkanjilal@yahoo.com

Joydip Kanjilal is an MVP (2007-2012), software architect, author, and speaker with more than 20 years of experience. He has more than 16 years of experience in Microsoft .NET and its related technologies. Joydip has authored eight books, more than 500 articles, and has reviewed more than a dozen books.



If you're to work with the code examples discussed in this article, you need the following installed in your system:

- Visual Studio 2022
- .NET 9.0
- ASP.NET 9.0 Runtime

If you don't already have Visual Studio 2022 installed on your computer, you can download it from here: <https://visualstudio.microsoft.com/downloads/>.

I'll start by showing you the basic concepts and then explore how you can program against each of them.

Overview of Threads, Multithreading, and Multitasking

In this section, I'll examine threads, multithreading, multitasking, multiprocessing, and other related concepts. Before I delve deeply into these concepts, let's understand what **resources** are in the context of an operating system.

Resources

Resources, in an operating system (OS), are components required by a process or thread to function. Typical examples of resources include hardware components, such as a central processing unit (CPU) time, memory space, input/output (I/O) devices (printers, disk drives), and files managed by the operating system to ensure that processes have access to these resources in a controlled and efficient manner. It's extremely important for an operating system to apply proper algorithms and strategies to manage and control the allocation, scheduling, and release of resources to ensure seamless functioning of operations.

From the operating system's perspective, there are two types of resources: preemptive resources and non-preemptive resources.

Preemptive resources can be taken away from a process and then re-allocated when the need arises. In other words, their allocation can be preemptively interrupted, and then assigned to another process, with the intention of reallocating back to the original process at a later time. The state of the resource can be saved and restored later, so the process can continue from where it was halted.

A good example of a preemptive resource is the CPU: An operating system can interrupt running processes (context switch), let other programs use the CPU for some share of time, and finally give it back to the first one without causing any harm. It should be noted that saving and restoring resource states of preemptive resources can be complex and costly.

Non-preemptive resources are also known as non-sharable resources because they remain with that process until they're released. Removing a non-preemptive resource from a process could either lead to processing errors or data corruption. Thus, no other process can use this resource until that particular process has finished with them and voluntarily releases them.

Typical examples of non-preemptive resources include printers and scanners, etc. A printer is a non-preemptive resource because once started, a printing job can't be stopped halfway through because multiple processes lead to possible loss or damage in output. The main problem associated with non-preemptive resources is avoiding resource deadlocks, because processes can't release their resources until they no longer require them.

CPU-Bound and I/O-Bound Operations

Any operation can be classified as either CPU-bound (or CPU-intensive) or I/O-bound (or I/O-intensive). A CPU-bound operation is one where the time spent on the CPU is greater. Long computations, such as massive data pro-

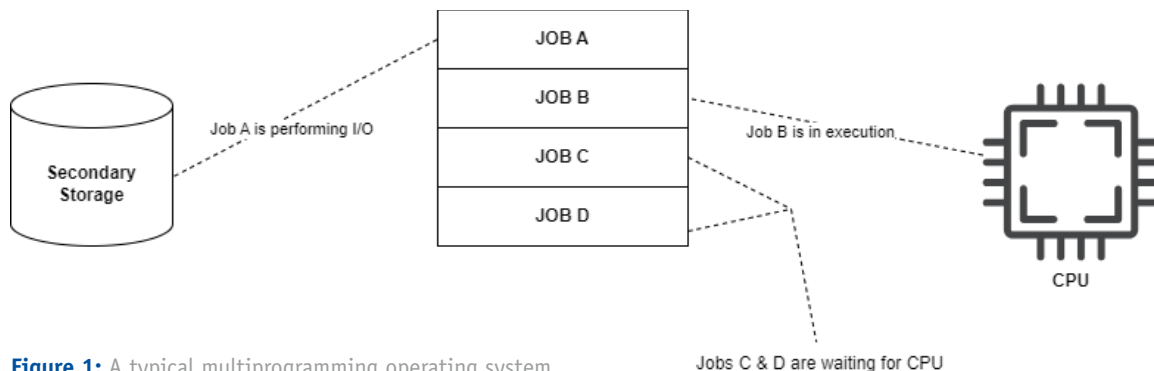


Figure 1: A typical multiprogramming operating system

cessing, are the most common examples. An I/O-bound operation is where the time spent performing I/O operations is greater. Typical examples are reading or writing files, copying files, reading or writing data to and from a database, and downloading data off the internet.

Process

A process is defined as an instance of a program in execution and is characterized by a change of state and attributes and identified by a Process Control Block (PCB) of its own. A process requires certain resources, including CPU, memory, files, and I/O devices, which are provided to it when the process is initially created or when it's in a running state. Each process contains its own memory space which, in turn, makes up the program's code, stack, and data. As soon as a running process terminates, it relinquishes control of all resources held and the operating system reclaims all reusable resources.

Thread

A thread is a low-level concept that represents the smallest unit of CPU use in a process. Memory space is shared between threads belonging to the same process. Threads are used to achieve parallelism on multi-core processors by enabling the execution of several threads concurrently. However, shared memory space in multi-threading is complicated because synchronization and state management become difficult to manage. It should be noted that any process must have at least one thread. This thread is also known as the main or primary thread. All other threads in a process are known as worker threads.

Task

A task is a high-level abstraction on top of a thread designed to support asynchronous operations. A task is often used in the context of asynchronous programming and encapsulates the job that needs to be performed. Unlike threads, which are managed by the underlying operating system, tasks are managed by the runtime environment in which they are executed, i.e., the CLR, if you're using .NET Core.

Multiprogramming

Basically, there are two types of processing: Serial Processing and Batch Processing. In serial processing, jobs are processed in a pre-determined sequence, meaning that each job is processed after the previous one has completed its execution. It should be noted that the processor is capable of executing one and only one job at a given point of time. Incidentally, serial processing was used before disk technology came into being.

Batch processing involves sending a bunch of programs to a tape drive at once. Typically, the jobs that have similar functionalities are grouped into a batch. These jobs are not processed at the same time. Instead, each request is processed one at a time, and the results are returned to the user after it has been processed. All such batches of jobs are read and run by the CPU and the result or output is written to another tape drive.

Multiprogramming is where an operating system loads multiple programs in the main memory (RAM) and runs them simultaneously. A variation of batch processing, multiprogramming attempts to maximize CPU use and enhances system throughput. This implies that it keeps the

CPU busy most of the time so that each program gets a fair share of the CPU processing time.

Figure 1 shows a typical multiprogramming operating system.

Multithreading

Typically, applications are either single-threaded or multithreaded based on how the processor executes the threads. In a single-threaded application, when the running thread has finished execution, the operating system schedules another thread for execution. This approach doesn't provide better system throughput (a measure of work completed per unit time). You cannot schedule another thread once a thread is in execution, and a single thread monopolizes the processor.

In a multithreaded application, several threads reside in the memory simultaneously, with one being in an execution state. Without waiting for the turnaround time of the currently executing thread to complete, new threads may be scheduled in these applications.

The CPU supports two types of scheduling: preemptive and non-preemptive. In preemptive scheduling, the thread in execution state is interrupted by the operating system and replaced by a different thread. This increases the system throughput because the threads waiting for the CPU are scheduled even before the executing thread relinquishes control of the processor. In non-preemptive scheduling, the running thread doesn't relinquish control of the CPU until its execution is complete.

The key benefits of multithreading include the following:

- Increased responsiveness
- Better resource sharing within the processes
- Enhanced scalability
- Better use of multiple processing cores

Figure 2 demonstrates a multithreaded application at work.

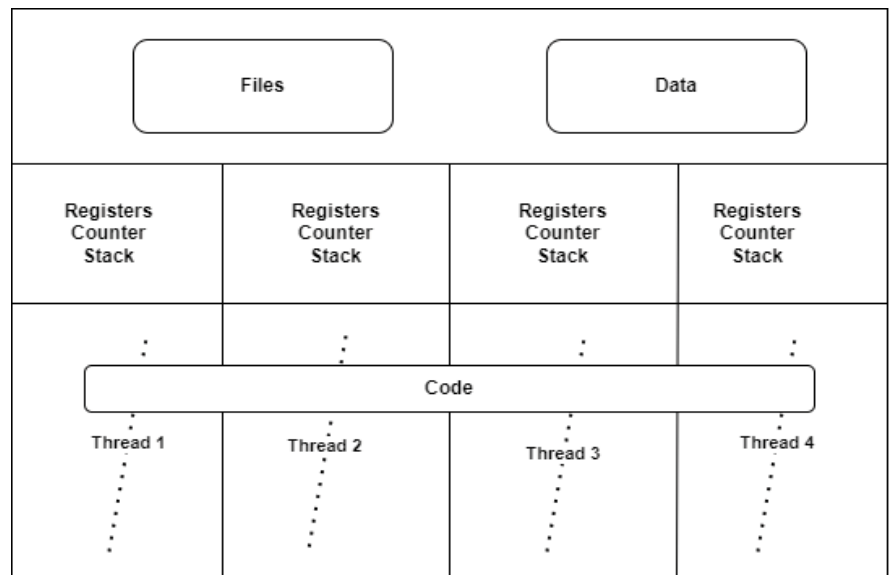


Figure 2: A typical multithreaded system

Multitasking

Multitasking is an operating system's capability to accomplish multiple tasks simultaneously by switching between them to enhance scalability and system throughput. In a typical multitasking system, the tasks are not actually executed simultaneously. Instead, the operating system uses its task scheduler to schedule tasks as needed and manages multiple tasks by allocating time slices to each of them.

Multitasking is of the following types:

- **Cooperative Multitasking:** This is a type of multitasking in which a running task voluntarily relinquishes control of the CPU to allow other tasks to run. It should be noted that Windows 3.x supported cooperative multitasking.

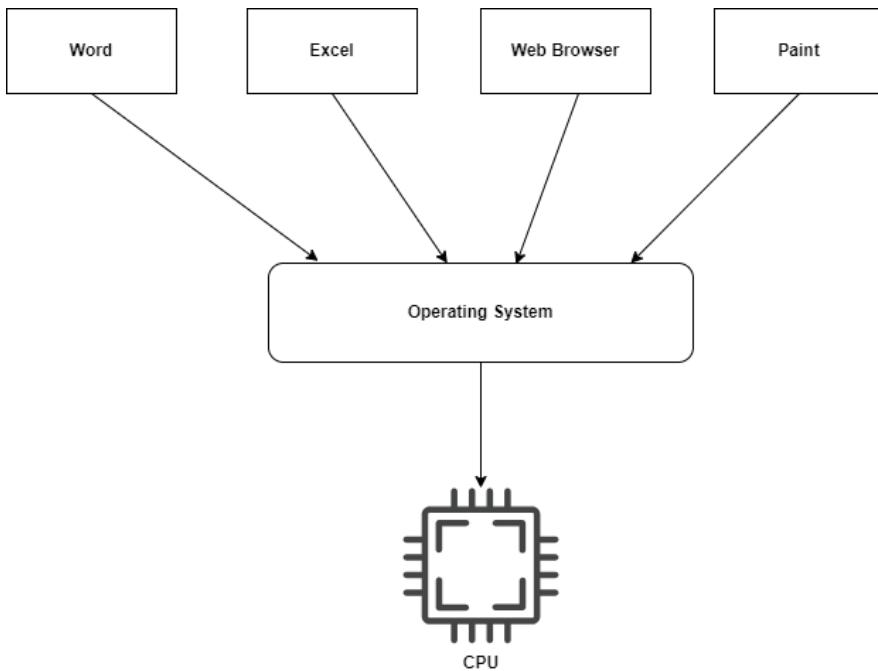


Figure 3: A typical multitasking system in action

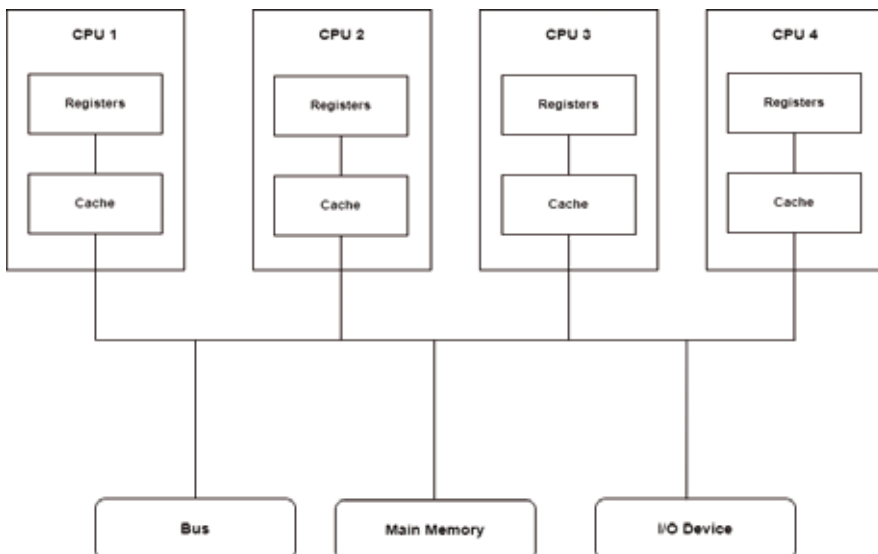


Figure 4: A typical multiprocessing system in action

- **Preemptive Multitasking:** In this case, the operating system allocates a limited time slice to each task, which allows the operating system to manage and control access to the CPU. As soon as this allotted time span elapses, the operating system preempts the running task and schedules another runnable task from the queue. This type of multitasking is supported by Windows 9x and later versions of the Windows operating systems.

Figure 3 shows a typical multitasking system at work.

Multiprocessing

Multiprocessing is defined as the ability of an operating system to run several programs or processes concurrently, either on a single processor with many cores or across multiple processors in the same system. This can improve performance, throughput and efficiency by allowing parallel execution of processes or threads across multiple CPUs or cores.

On a single core system, the threads pertaining to a multithreaded application never execute in parallel because they have to share a single core. On a multi-core system, there are multiple cores within a single processor and you can execute tasks in parallel. Figure 4 shows a multiprocessor system with multiple cores.

Programming Threads in C#

In C#, the Thread class pertaining to the System.Threading namespace works with threads. Remember, you can have two types of threads: an application thread and one or more worker threads. Although the application thread is created by an application automatically by the runtime, any thread you create programmatically is a worker thread.

The Thread class provides three important properties: IsAlive, IsBackground, and IsThreadPoolThread. You may check whether the thread is alive using the read-only IsAlive property of the Thread class. If the thread is alive, it contains a value true, and false otherwise. The IsBackground property contains a value true if the thread is a background thread, and false otherwise.

When set to true, this property transforms a thread into a background thread. The IsThreadPoolThread is yet another read-only property that indicates if the thread belongs to the managed thread pool. If it's a thread pool thread, this property contains a value true, and false otherwise.

Creating a Thread

When a thread is first created, it's in a dormant state. To start a thread in C#, make a call to the Start method of the Thread class. The following code snippet demonstrates how you can create a worker thread in C#.

```
static void ThreadMethod()
{
    Console.WriteLine("Worker thread.");
}

static void CreateThread()
{
    Console.WriteLine("Primary thread.");
    Thread threadObj = new Thread(new
        ThreadStart(ThreadMethod));
```

```
threadObj.Start();
}
```

In the preceding code example, as the name suggests, the method named `ThreadMethod` is the method that gets called when the thread is started. Note that `ThreadStart` is a delegate used to represent the thread method that gets executed when a thread is started. The following code snippet shows how you can start a worker thread which, in turn, is attached to a method that displays integers 1 to 10.

```
static void CreateThread()
{
    Thread threadObj = new Thread(new
        ThreadStart(DisplayNumbers));
    threadObj.Start();
}

static void DisplayNumbers()
{
    for (int i = 1; i <= 10; i++)
    {
        Console.WriteLine(i);
    }
}
```

You can also create a thread by passing the name of the thread method directly in the constructor of the `Thread` class, as shown in the following code example:

```
static void CreateThread()
{
    Console.WriteLine("Primary thread.");
    Thread threadObject =
        new Thread(ThreadMethod);
    threadObject.Start();
}

static void ThreadMethod()
{
    Console.WriteLine("Worker thread.");
}
```

Because creation and disposal of threads are resource-intensive operations, you can use a thread from the managed thread pool to queue tasks without having to manage the threads individually. The following code snippet illustrates how this can be achieved.

```
ThreadPool.QueueUserWorkItem
((state) => {
    Console.WriteLine
("This code will be executed
by a thread pertaining
to the managed thread pool.");
});
```

Putting a Thread to Sleep in C#

You can leverage the `Thread.Sleep()` method to put a running thread to sleep. The `Thread.Sleep` method accepts an integer as a parameter that represents the timeout in milliseconds, as shown in the code snippet below.

```
static void CreateThread()
{
    Thread threadObject =
        new Thread(ThreadMethod);
    threadObject.Start();
    Thread.Sleep(1000);
}
```

When you invoke the `Thread.Sleep` method, it instantly suspends the thread that called it for the period of time specified in milliseconds as a parameter. You can ensure that a thread runs indefinitely by using `while(true)` in your thread method. This consumes CPU cycles and is resource intensive. A better alternative is using `Timeout.Infinite` as a parameter to the `Thread.Sleep` method, as shown below.

```
Thread.Sleep(Timeout.Infinite);
```

A call to the `Thread.Sleep` method puts the current execution context to sleep by invoking the sleep function of the OS kernel, allowing the CPU to continue to do other work.

Setting Thread Priority

The `Thread` class provides an enum called `ThreadPriority` that contains all supported thread priorities: `Lowest`, `BelowNormal`, `Normal`, `AboveNormal`, and `Highest`. By default, a managed thread is created with a thread priority `ThreadPriority.Normal`. You can also set the priority of a managed thread when creating it using the following piece of code:

```
static void SetThreadPriority()
{
    Thread threadObj = new Thread(new
        ThreadStart(DisplayNumbers));
    threadObj.Priority =
        ThreadPriority.AboveNormal;
    threadObj.Name = "SetThreadPriority";
    threadObj.Start();
}

static void DisplayNumbers()
{
    for (int i = 1; i <= 10; i++)
    {
        Console.WriteLine(i);
    }
}
```

Retrieving the State of a Thread

You can take advantage of the `ThreadState` property to get the state of a thread, as shown in the code snippet below:

```
static void CreateThread()
{
    Console.WriteLine("Primary thread.");
    Thread threadObject =
        new Thread(ThreadMethod);
    Console.WriteLine
("The thread state is: " +
    threadObject.ThreadState);
    threadObject.Start();
    Thread.Sleep(1000);
    Console.WriteLine
```

```

        ("The thread state is: " +
        threadObject.ThreadState);
    }

    static void ThreadMethod()
    {
        Console.WriteLine("Worker thread.");
    }

```

All supported states of a managed thread are defined in the `ThreadState` Enum. You can take advantage of the public read-only property called `ThreadState` to know the state of a particular managed thread.

Suspending and Resuming a Thread

In C#, a thread can be suspended or resumed using the deprecated methods `Suspend` and `Resume`. These methods are deprecated and their use is discouraged in .NET Framework and .NET Core. This is because if you suspend a thread that's already inside a critical section where it's holding a lock on a critical resource, the entire application might deadlock. A better way to handle this is by using `WaitHandle`.

`WaitHandlers` help threads communicate with one another using signaling where a particular thread waits until it receives a notification from another thread. In C#, you can have two types that represent `EventWaitHandlers`, `AutoResetEvent` and `ManualResetEvent`. The basic difference between an `AutoResetEvent` and a `ManualResetEvent` is that an `AutoResetEvent` only allows one waiting thread to continue, and a `ManualResetEvent` allows multiple threads to continue until you stop it.

In `AutoResetEvent`, `WaitOne()`, and `Reset()` are executed as a single atomic operation. To be more precise, although `AutoResetEvent` enables one of the waiting threads to pass when the `Set()` method is called, the latter allows all waiting threads to pass when the `Set()` method is called. In addition, the `AutoResetEvent` releases only one waiting thread at a time. It should be noted that an `AutoResetEvent` resets automatically when your code passes through `eventObj.WaitOne()` and a `ManualResetEvent` does not.

Terminating a Thread

The `Thread.Abort()` method can be used to abort a running thread. The following code snippet shows how the `Thread.Abort` method can be used in C#:

```

Thread threadObj = new Thread(PerformSomeWork);
threadObj.Start();
//Usual code
threadObj.Abort();

```

However, this isn't a recommended approach and has been deprecated in .NET Core because it adopts an unsafe approach to terminating threads. A recommended approach to thread termination is by using `CancellationToken`, as shown in the code snippet given below:

```

public static void Start()
{
    CancellationTokenSource cancellationToken =
    new CancellationTokenSource();
    Thread thread = new Thread(() =>
    MyThreadMethod(cancellationToken.Token));

```

```

        thread.Start();
        Thread.Sleep(1000);
        cancellationToken.Cancel();
        thread.Join();
    }

    public static void
    MyThreadMethod(CancellationToken
    cancellationToken)
    {
        while (!cancellationToken.
        IsCancellationRequested)
        {
            Thread.Sleep(1000);
            Console.WriteLine
            ("The thread method is running...");
        }

        Console.WriteLine
        ("Cancellation requested, exiting thread.");
    }

```

Deadlock

In a multiprogramming environment, several processes may often vie for a limited number of resources at the same time. When a process needs access to any resource but it isn't available at the moment, it enters a waiting state and a deadlock occurs. A deadlock is when two or more processes (or threads) wait for one another to release a resource or multiple resources that create cyclic dependencies and none of them are allowed to continue further. All of those processes are also waiting, yet they cause others to wait because each of them is awaiting fulfillment of a condition that only exists in another process.

Let's say that process P1 and process P2 have acquired resources R1 and R2 respectively. Process P1 enters into a waiting state when process P2 hasn't relinquished control of resource R2. Process P2 enters a waiting state to acquire control of resource R1 that has been acquired by process P1. The result is deadlock because one process (say, P1) waits indefinitely for access to shared resources that has already been acquired by another process (say, P2), thereby becoming entangled in a vicious cycle.

The following code snippet illustrates the entire process:

```

lock(R1)
{
    //Some code
    lock(R2)
    {
        //Some code
    }
}

lock(R2)
{
    //Some code
    lock(R1)
    {
        //Some code
    }
}

```

The following piece of code shows how to use the `Monitor.TryEnter` method to acquire an exclusive lock on a shared object.

```
public class DeadlockExample
{
    private readonly static object
    lockObj = new object();

    public static void PreventDeadlock
    (object sharedInstance)
    {
        try
        {
            if (Monitor.TryEnter(lockObj,
                TimeSpan.FromMilliseconds(10)))
            {
                //This is the critical section
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }
        finally
        {
            if (Monitor.IsEntered
                (sharedInstance))
                Monitor.Exit(sharedInstance);
            Monitor.Exit(lockObj);
        }
    }
}
```

The complete source code of the `DeadlockExample` class is given in **Listing 1**. In **Listing 1**, `ProcessA` and `ProcessB` acquire locks on resources `objA` and `objB` respectively. `ProcessA` waits to acquire access on resource `objB` after acquiring `objA`, and `ProcessB` waits for access to resource `objA` after acquiring access to resource `objB`, resulting a deadlock situation.

In C#, several methods of the classes pertaining to the `System.Threading` namespace provide support for time-outs to help you determine deadlocks. For example, the following code snippet shows how you can gain a lock on a shared resource within a specific time interval. If it's not possible

to gain the lock on the shared resource within the specified time frame, the `Monitor.TryEnter` method returns false.

```
if (Monitor.TryEnter(lockObj, 10))
{
    try
    {
        //This is the critical section.
        //Write your code here.
    }
    finally
    {
        Monitor.Exit(lockObj);
    }
}
else
{
    //This code will execute if the
    //request for acquiring a lock
    //on the shared resource times out.
}
```

Listing 2 shows how you can avoid the deadlock situation by establishing a consistent locking order, ensuring that the processes acquire locks on the resources in a consistent order to avoid any potential circular dependency. The `ExecuteProcessA()` method contains the necessary source code for process A, the `ExecuteProcessB()` method includes the source code for process B. Note how circular dependencies have been prevented in these two methods by using a lock hierarchy, i.e., using nested locks. You can ensure that there will be no circular wait using lock hierarchy because the locks are acquired in the same order by processes A and B. You can prevent deadlock in this manner.

Asynchronous Programming

Asynchrony is a programming paradigm that allows tasks to execute independently of the primary or the main thread of execution, thereby enabling programs to proceed with other tasks while waiting for an operation to complete. You can take advantage of callbacks, promises, futures, or events to handle completion of an asynchronous task.

.NET Core provides support for executing a piece of code or a method in a synchronous or asynchronous manner.

Listing 1: Demonstrating a potential deadlock situation

```
static void ExecuteProcessA()
{
    lock (objA)
    {
        Console.WriteLine
        ("Process A is holding lock on resource objA");
        Thread.Sleep(1000);

        Console.WriteLine
        ("Process A is waiting for resource objB");
        lock (objB)
        {
            Console.WriteLine
            ("Process A is holding lock on objB");
        }
    }
}
```

```
static void ExecuteProcessB()
{
    lock (objB)
    {
        Console.WriteLine
        ("Process B is holding lock on resource objB");
        Thread.Sleep(1000);

        Console.WriteLine("Process B is waiting for resource objA");
        lock (objA)
        {
            Console.WriteLine("Process B is holding lock on objA");
        }
    }
}
```


Listing 2: Solution to Deadlock by avoiding circular dependencies

```
static void ExecuteProcessA()
{
    lock (objA)
    {
        Console.WriteLine
            ("Process A is holding lock on resource objA");
        Thread.Sleep(1000);

        Console.WriteLine
            ("Process A is waiting for resource objB");
        lock (objB)
        {
            Console.WriteLine
                ("Process A is holding lock on objB");
        }
    }
}

static void ExecuteProcessB()
{
    lock (objA)
    {
        Console.WriteLine
            ("Process B is holding lock on resource objA");
        Thread.Sleep(1000);

        Console.WriteLine
            ("Process B is waiting for resource objB");
        lock (objB)
        {
            Console.WriteLine
                ("Process B is holding lock on objB");
        }
    }
}
```

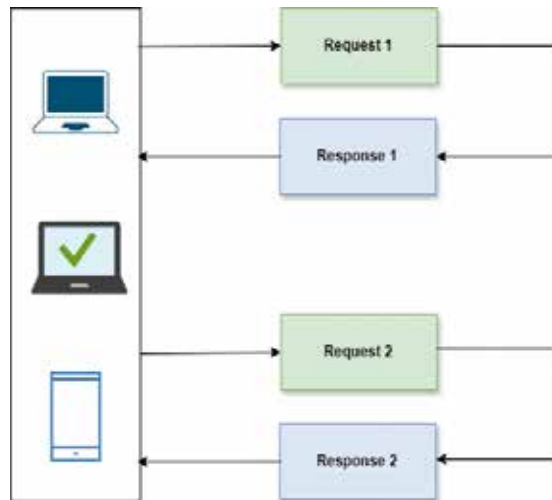


Figure 5: A typical synchronous execution model

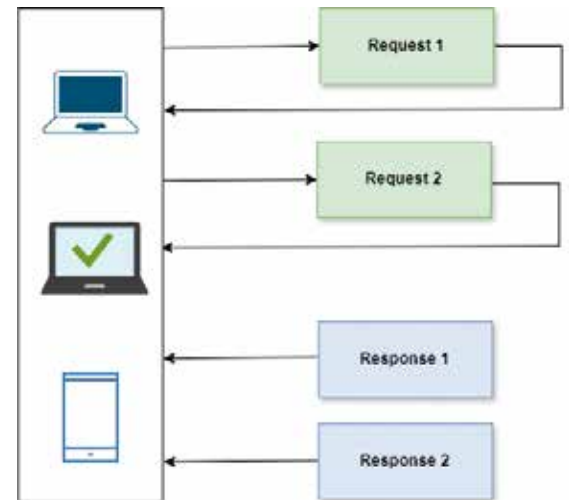


Figure 6: A typical asynchronous execution model

Synchronous execution is a programming model that is blocking in nature. This implies that the execution of the program halts at each statement and waits for the statement to terminate before executing the next statement. As a result, the program pauses at each statement, resulting in execution delays. **Figure 5** shows how a synchronous execution model works.

Asynchronous programming refers to a programming paradigm where the tasks can run independently from one another. This ensures high throughput and improved responsiveness for your application. In other words, with asynchronous programming, you can execute tasks without blocking the execution flow or responsiveness of the application. By using asynchronous programming, you can make your application more responsive, scalable, and performant. **Figure 6** shows how an asynchronous execution model works.

Why Asynchrony Is Important

A non-blocking or asynchronous execution allows for executing tasks without waiting or slowing down the execution flow of your application.

Here are the benefits of asynchronous program at a glance:

- It makes the user interface responsive.
- It increases scalability and performance of applications.
- It prevents thread pool starvation.

There are certain downsides as well:

- Asynchronous code is complex and difficult to maintain.
- Asynchronous code might consume additional resources (memory, CPU, etc.).
- Debugging asynchronous programming can be extremely challenging.

The *async* and *await* Keywords

The **async** keyword converts a method into an asynchronous one enabling the use of the **await** keyword inside the method body. Note that `Task` and `Task<T>` are two awaitable types provided by .NET Core. An async method returns a `Task` or `Task<T>` object and performs non-block-

Listing 3: The complete source of the DeadlockExample class

```
public static class DeadlockExample
{
    static readonly object objA = new object();
    static readonly object objB = new object();

    public static void DeadlockDemo()
    {
        Thread threadA = new Thread(ExecuteProcessA);
        Thread threadB = new Thread(ExecuteProcessB);

        threadA.Start();
        threadB.Start();

        threadA.Join();
        threadB.Join();

        Console.WriteLine("End of program");
    }

    static void ExecuteProcessA()
    {
        lock (objA)
        {
            Console.WriteLine
                ("Process A is holding lock on resource objA");
            Thread.Sleep(1000);

            Console.WriteLine
                ("Process A is waiting for resource objB");

            lock (objB)
            {
                Console.WriteLine
                    ("Process A is holding lock on objB");
            }
        }
    }

    static void ExecuteProcessB()
    {
        lock (objA)
        {
            Console.WriteLine
                ("Process B is holding lock on resource objA");
            Thread.Sleep(1000);

            Console.WriteLine
                ("Process B is waiting for resource objB");
            lock (objB)
            {
                Console.WriteLine
                    ("Process B is holding lock on objB");
            }
        }
    }
}
```

ing operations. You must use the await keyword to call an async method.

Some key features of the async keyword include:

- You can use the async keyword for methods, lambda expressions, and anonymous methods, but not for constructors or properties.
- An async method should have a minimum of one await expression in its body. In an async method, you can have multiple await keywords to handle multiple non-blocking operations.
- Methods that are async can be chained together.

Using the await keyword suspends the calling method until the async task finishes and returns control to the caller. In other words, when the await keyword is used to invoke an asynchronous method, it pauses execution of the awaited method and asynchronously waits for the Task to complete while the currently executing thread is sent back to the thread pool in lieu of keeping it in a blocked state. It should be noted that the await keyword can be used only in an async method. The caller method, i.e., the method that invokes the async method using the await keyword, should also be made asynchronous.

Here are the key features of the await keyword:

- You can use the await keyword only inside an async method.
- It's possible to use the await keyword in any expression that returns a Task or a Task object.
- When the await keyword is used, the result of the Task object is unwrapped. This helps you to work directly with the result of the Task.
- Usage of the await keyword causes the calling

method to be paused and relinquish the control back to the caller method until the awaited task is finished.

Consider the following piece of code that demonstrates an asynchronous method:

```
public async Task MyAsyncMethod()
{
    await Task.Delay(100);
}
```

You can invoke this method asynchronously using this code snippet:

```
await MyAsyncMethod();
```

In C#, you can define async void methods by preceding the async before the method signature, followed by a void return type. The following code snippet shows how this can be achieved:

```
public async void AnAsyncVoidMethod()
{
    //Write your code here
}
```

CPU-Bound and I/O-Bound Code

You can use asynchronous code for CPU-bound as well as I/O-bound code but in different ways. You typically need to leverage Task and Task<T> in async code, which are constructs used to model work done in the background. When a method has the async keyword, it's converted into an asynchronous method, which allows you to include the await keyword within the body of the method. Note that

Listing 4: Async File Operations

```
using System.IO;
using System.Text;

string sourceFileName = @"D:\Input\SampleData.txt";
string destinationFileName = @"D:\Output\SampleData.txt";

string text = "Hello World!";
await WriteFileAsync(@"D:\SampleData.txt", text);

await CopyToAsync(sourceFileName, destinationFileName);

var data = await ReadFileAsync(sourceFileName);
Console.WriteLine(data);

Console.Read();
async Task<string> ReadFileAsync(string inputFileName)
{
    using (var streamReader =
        new StreamReader(inputFileName))
    {
        return await streamReader.ReadToEndAsync();
    }
}

async Task WriteFileAsync
(string destinationFileName, string text)
{
    byte[] buffer = Encoding.Unicode.GetBytes(text);
    int offset = 0;
    const int Buffer_Size = 4096;

    bool isAsync = true;

    using (var fileStream = new FileStream
        (destinationFileName, FileMode.Append,
        FileAccess.Write, FileShare.None,
        Buffer_Size, isAsync))
    {
        await fileStream.WriteAsync
            (buffer, offset, buffer.Length);
    }
}

async Task CopyToAsync
(string sourceFileName,
string destinationFileName)
{
    using (FileStream fileStream =
        File.Open(sourceFileName, FileMode.Open))
    {
        using (FileStream destination =
            File.Create(destinationFileName))
        {
            await fileStream.
                CopyToAsync(destination);
        }
    }
}
```

unlike I/O-bound tasks, CPU-bound tasks require a worker thread to operate. These tasks leverage a thread from the thread pool to function.

The following method illustrates how you can implement CPU-bound async method in C#.

```
public async Task<int>
GenerateFactorialAsync(int number)
{
    int factorial = 1;
    for (int i = 1; i <= number; i++)
    {
        factorial *= i;
    }

    return await
        Task.FromResult(factorial);
}
```

You can use the following piece of code to invoke the GneerateFactorialAsync async method you just created.

```
int number = 5;
int result =
await Task.Run(() =>
    GenerateFactorialAsync(number));
Console.WriteLine
($"The factorial of {number}
is: {result}");
```

The asynchronous programming model uses the `async` and the `await` keywords to implement asynchronous operations. In this programming paradigm you should:

- Await an asynchronous operation that returns a `Task` or a `Task<T>` object for I/O-bound code.
- Await an asynchronous operation initiated on a background thread by invoking the `Task.Run()` method for CPU-bound code.

The following code snippet is an example of an I/O-bound operation that shows how you can use `await` keyword to retrieve the content of a webpage asynchronously:

```
HttpClient httpClient =
new HttpClient();
HttpStatusCode status;
HttpResponseMessage
httpResponseMessage =
await httpClient.GetAsync
("https://www.joydipkanjilal.com");
string content =
await httpResponseMessage.
Content.ReadAsStringAsync();
Console.WriteLine(content);
```

Asynchronous File Operations

Create two string variables that contain the file name of path of the input file and the destination file respectively. The following code snippet shows the `ReadFileAsync` method that can read a file asynchronously.

```
public async Task<string>
ReadFileAsync(string inputFileName)
{
    using (var streamReader = new
        StreamReader(inputFileName))
    {
        return await
            streamReader.ReadToEndAsync();
    }
}
```

```
}
}
```

You can now invoke the `ReadFileAsync` method using the following piece of code:

```
var data = await ReadFileAsync
(@"D:\SampleData.txt");
Console.WriteLine(data);
```

To copy files asynchronously, you can use the `CopyToAsync` method of the `FileStream` class as shown below:

```
public async Task CopyToAsync
(string inputFileName,
string destinationFileName)
{
    using (FileStream fileStream =
        File.Open(inputFileName,
```

```
FileMode.Open))
{
    using (FileStream destination =
        File.Create(destinationFileName))
    {
        await fileStream.
            CopyToAsync(destination);
    }
}
```

The following code snippet shows how you can write a file asynchronously:

```
public async Task
WriteFileAsync
(string destinationFileName,
string text)
{
    byte[] buffer =
```

Listing 5: The `RetrieveAllExceptionsAsync` method

```
public async Task RetrieveAllExceptionsAsync()
{
    var firstTask = Task.Run(() =>
    throw new ArithmeticException
    ("Error occurred: " +
    typeof(ArithmeticException).ToString()));

    var secondTask = Task.Run(() =>
    throw new IndexOutOfRangeException
    ("Error occurred: " +
    typeof(IndexOutOfRangeException).ToString()));

    var thirdTask = Task.Run(() =>
    throw new InvalidOperationException
    ("Error occurred: " +
    typeof(InvalidOperationException).ToString()));

    var fourthTask = Task.Run(() =>
    throw new DivideByZeroException
    ("Error occurred: " +
    typeof(DivideByZeroException).ToString()));

    var fifthTask = Task.Run(() =>
    throw new IOException
    ("Error occurred: " +
    typeof(IOException).ToString()));

    Task tasks = Task.WhenAll
    (firstTask, secondTask,
    thirdTask, fourthTask, fifthTask);

    try
    {
        await tasks;
    }
    catch
    {
        AggregateException exceptions =
            tasks.Exception;
    }
}
```

Listing 6: The `MultipleExceptionsAsync` method

```
public async Task MultipleExceptionsAsync()
{
    Task tasks = null;
    try
    {
        var firstTask = Task.Run(() =>
        throw new
        ArithmeticException
        (typeof(ArithmeticException).
        ToString()));
        var secondTask = Task.Run(() =>
        throw new
        IndexOutOfRangeException
        (typeof(IndexOutOfRangeException).
        ToString()));
        var thirdTask = Task.Run(() =>
        throw new
        InvalidOperationException
        (typeof(InvalidOperationException).
        ToString()));

        tasks = Task.WhenAll
        (firstTask, secondTask, thirdTask);
        await tasks;
    }
    catch
    {
        AggregateException exceptions =
            tasks.Exception;

        foreach (var ex in tasks.Exception?.
            InnerExceptions ??
            new(Array.Empty<Exception>()))
        {
            Console.WriteLine
            (ex.GetType().ToString());
        }
    }
}
```

```

Encoding.Unicode.GetBytes(text);
int offset = 0;
const int Buffer_Size = 4096;
bool isAsync = true;

using (var fileStream = new FileStream
(destinationFileName, FileMode.Append,
FileAccess.Write, FileShare.None,
Buffer_Size, isAsync))
{
    await fileStream.WriteAsync
(buffer, offset, buffer.Length);
}
}

```

The complete source code is given in **Listing 4**.

Handling Exceptions

Synchronous and asynchronous methods handle exceptions differently. In a synchronous method, if the runtime raises an exception, the exception object is sent up the call stack until it hits a suitable catch block capable of handling the instance.

If an exception is thrown in an async method, it's stored in the task object that's returned by the method. These exceptions aren't visible until the task is awaited. Notably, asynchronous methods with a return type of **void** lack an accompanying Task object. If exceptions occur in these methods, they're triggered on the SynchronizationContext that was in use when the asynchronous method was invoked. **Listing 5** shows the RetrieveAllException-Async method that illustrates how exceptions can be retrieved from a Task instance.

Whenever an async method throws an exception, the exception object gets wrapped in an AggregateException instance. Note that you can retrieve all exceptions thrown inside an async method using the Exceptions property of the Task object. **Listing 6** shows how you can

handle multiple exceptions thrown in an asynchronous method.

The mechanism of handling exceptions in async void and async Task methods differ. The primary difference between async void and async Task methods depends on how exceptions are handled in them. When exceptions occur in an async Task method, the exceptions are captured by the Task object returned by the method. This enables you to handle the exception or await the Task and handle the exceptions later. On the other hand, async void methods don't have any Task object. You cannot await async void methods. Hence, any exceptions that occur inside these methods are propagated up to the SynchronizationContext that started the async method initially.

Consider the following code snippet that shows two methods, ProcessData and ProcessDataAsync. The former handles the exception and the latter throws an exception after a delay of 100 milliseconds.

```

public async void ProcessData()
{
    try
    {
        ProcessDataAsync();
    }
    catch (Exception ex)
    {
        Console.WriteLine
("Error: " + ex.Message);
    }
}

public async void ProcessDataAsync()
{
    await Task.Delay(100);
    throw new Exception
("This is a test error message.");
}

```

Listing 7: The MultipleExceptionsInAsyncCodeDemo method

```

public async Task
MultipleExceptionsInAsyncCodeDemo()
{
    try
    {
        var firstTask = Task.Run(() =>
            throw new
            ArithmeticException
            ("Error occurred: " +
            typeof(ArithmeticException).ToString());
        var secondTask = Task.Run(() =>
            throw new
            IndexOutOfRangeException
            ("Error occurred:
            "+typeof(IndexOutOfRangeException).
            ToString());
        var thirdTask = Task.Run(() =>
            throw new
            InvalidOperationException
            ("Error occurred: " +
            typeof(InvalidOperationException).
            ToString());

        Task.WaitAll
            (firstTask, secondTask, thirdTask);
    }
    catch (AggregateException ae)
    {
        ae.Handle(ex =>
        {
            if (ex is IndexOutOfRangeException)
                Console.WriteLine
                ("An exception of type
                IndexOutOfRangeException occurred: " +
                ex.Message);

            if (ex is InvalidOperationException)
                Console.WriteLine
                ("An exception of type
                InvalidOperationException occurred: " +
                ex.Message);

            return ex is
                InvalidOperationException;
        });
    }
}

```


When you run the above piece of code, the exception thrown inside the `ProcessDataAsync` method isn't caught inside the catch block of the caller method, the `ProcessData` method. To solve this, change the signature of the `ProcessDataAsync` method from `async void` to `async Task` as shown in the code snippet given below.

```
public async Task ProcessDataAsync()
{
    await Task.Delay(100);
    throw new Exception
        ("This is a test error message.");
}

public async void ProcessData()
{
    try
    {
        await ProcessDataAsync();
    }
    catch (Exception ex)
    {
        Console.WriteLine
            ("Error: " + ex.Message);
    }
}
```

Note that you can also leverage `AggregateException`. Handle to handle exceptions that you'd like to handle and ignore those you don't want to handle, as shown in **Listing 7**. Note how the `IndexOutOfRangeException` and `InvalidOperationException` is handled while the `ArithmeticException` is ignored.

Best Practices

Here are some of the best practices you should adhere to when working with asynchronous methods in C#:

- Mark methods having one `await` expression with the `async` keyword so that the method signature clearly shows that such methods are asynchronous.
- In asynchronous methods, return a `Task` or `Task<T>` rather than `void`, so that the caller method can await the results and handle errors elegantly.
- Don't use `async void` methods because they can't be awaited and can potentially trigger unhandled exceptions. Instead, take advantage of `async Task` methods for implementing event handlers. This provides support for proper exception handling, enabling you to handle exceptions elegantly.
- To reduce deadlocks, use `ConfigureAwait(false)` to reduce the likelihood of capturing the context.
- Leverage `Task.Run` for CPU-bound operations in your application. This helps offload the work onto a separate thread and prevents blocking the main or the primary thread.
- Take advantage of `SemaphoreSlim` or `Task.WhenAll` to throttle the maximum number of concurrent tasks when you call an asynchronous method inside a loop to avoid increased usage of system resources.

Parallelism

In computing, parallelism is defined as the capability of an operating system to accomplish multiple tasks con-

currently by leveraging multiple processors or cores. You can take advantage of parallelism to increase the overall performance and responsiveness of your applications that process large amounts of data or handle computationally intensive tasks.

Why Parallelism Is Important

By leveraging all the of the CPUs present in multi-core PCs, and high-performance computing clusters (HPS clusters), parallel programming becomes inevitable when handling problems that involve massive computations. In parallel programming, you decompose a problem into smaller workable steps that can be executed concurrently. In parallel programming, the tasks are parallelized during their execution so that they can run simultaneously on different cores within a CPU. Parallel programming is useful where there is massive amount of data and application performance is crucial.

Although parallelism alludes to greater benefits in performance and scalability, it's not a panacea. Not all queries should be parallelized, particularly when interdependencies are involved. For example, one task might need to wait for another to complete before it can move ahead. In this case, parallelizing your tasks becomes quite a challenge.

Data and Task Parallelism

Data parallelism and task parallelism are two approaches to parallel computation often used to improve application performance on multi-core processors by splitting a problem into subtasks that can be solved at the same time. Additionally, the Task Parallel Library (TPL) supports data-parallel operations, such as executing loops in parallel (`Parallel.For` and `Parallel.ForEach`).

Data parallelism involves dividing data into subsets and performing identical operations on each subset at the same point of time. This approach is very effective for numerical algorithms that operate intensively on large datasets by repeatedly performing identical steps over elements.

In image processing, you use data parallelism if you need to go through every pixel in an image (like applying a filter), or for simulations where calculations have to be done again for lots of data points. Note that big data and other distributed systems need data parallelism because it enables developers to solve problems faster using more machines.

Unlike data parallelism, which entails performing the same operation on different data sets, **task parallelism** is concerned with performing multiple operations on a single dataset. When you do task parallelism, you break a problem into separate parts, called tasks, which can be executed concurrently. Leverage this type of parallelism when tasks involve complex operations that might not operate on the same dataset.

In task parallelism, each task does a different thing and operates on different datasets. These operations may involve doing the same work with various pieces of information or doing totally different things using them. This type of parallelism is appropriate when the tasks are quite separate and complex and where each one operates on its own dataset.

Parallel Programming

Parallel programming or parallel processing is a type of computation where several computations are all performed at once. This can help you solve a problem faster, generally by splitting the work across multiple processors or across multiple cores within the CPU. This technique incorporates modern computers that possess multiple CPUs or cores and can produce high throughput in less time.

Here are the key use cases of parallel programming:

- Climate research
- Applied physics
- Quantum mechanics
- Advanced graphics
- Modular modeling

Parallel Extensions

Parallel Extensions, formerly known as Parallel Framework Extensions (PFX), is an open-source, lightweight, managed concurrency library. With it, you can support both imperative and declarative parallelism as well as data and task parallelism. This allows LINQ developers to take advantage of multi-core systems while having complete support for all .NET query operators with minimal impact on the existing LINQ model. The library includes APIs that enable you to use multiple cores in your system and implement data parallelism and task parallelism in your applications.

The following libraries are a part of PFX:

- Task Parallel Library (TPL)
- Parallel LINQ (PLINQ)

Task Parallel Library (TPL)

The Task Parallel Library (TPL) encompasses a collection of APIs pertaining to the System.Threading and System.Threading.Tasks namespaces that are used to implement parallelism and concurrency. It provides an abstraction over lower-level constructs like threads, synchronization primitives, etc., allowing you to write more scalable code without having to deal with these low-level details directly.

The TPL includes the Parallel class, located in the System.Threading namespace. This class provides static methods, such as For, ForEach, and Invoke. You should leverage Parallel.For and Parallel.ForEach methods to parallelize loops or achieve imperative data parallelism. Use Parallel.Invoke method to implement task parallelism.

The TPL is available in the System.Threading.Tasks namespace and is used to work with Task and Task<T> types. TPL abstracts several complexities associated with multi-threading that include creation of threads, synchronization, and thread pooling.

A Task is the basic unit of work in the TPL, represented as a single asynchronous operation analogous to a thread or a ThreadPool work item at a higher level of abstraction. A task can be executed concurrently with other tasks and can be created using the Task class or by invoking the Task.Run() method explicitly.

The following code snippet shows how you can create a basic Task in C#:

```
Task task = Task.Run(() =>
{
    Console.WriteLine
        ("Demonstrating a basic Task in execution");
});

task.Wait();
```

Note that you should use Task<T> when you need to return a value. Here, T refers to the type of the value that should be returned from the async method. The following code snippet shows how you can return a value from a task in C#.

```
public static async Task Start()
{
    int x = 5, y = 10;
    Task<int> task = MultiplyAsync(x, y);
    int result = await task;
    Console.WriteLine($"The result is: {result}");
}

public static async Task<int>
    MultiplyAsync(int x, int y)
{
    await Task.Delay(100);
    return x * y;
}
```

The following piece of code shows how you can leverage the AsParallel() method to store the prime numbers between 1 and 100 in a collection, while handling the exceptions (if any) thrown in the try block.

```
try
{
    int[] integers =
        Enumerable.Range(1, 100).ToArray();

    var result = integers
        .AsParallel()
        .Select(
            n =>
            {
                if (n % 100 == 0)
                    throw new
                        InvalidOperationException(
                            $"Error occurred when the
                                value of n was {n}"
                        );
                return n;
            }
        )
        .Where(n => IsPrimeNumber(n))
        .ToList();
}
catch (AggregateException ae)
{
    foreach (var ex in ae.InnerExceptions)
    {
        Console.WriteLine
            ($"Exception occurred: {ex.Message}");
    }
}
```

Parallel LINQ (PLINQ)

Parallel LINQ, generally referred to as PLINQ, is a query execution engine for .NET Framework and .NET Core framework applications that runs LINQ to an object or LINQ to XML queries on multiple processors and cores. It's a part of the Parallel Extensions Library (also known as Parallel Framework Extensions—PFX—previously) that splits the input data into partitions that can be handled independently by one or more threads.

Under the hood, PLINQ uses partitioning. This means that the input data set is divided into chunks/blocks/data partitions (these are in-memory). Each processing core leverages its own partitioned subset of the total collection being processed. Once all these results have been obtained individually from each thread running at different CPU cores simultaneously, the data is merged. This merging process involves combining the results from each thread to form the final set of data.

The following code snippet illustrates a basic PLINQ query:

```
var integers =
    Enumerable.Range(1, 100);
var result = integers.AsParallel().
    Where(n => n % 2 == 0).ToList();

Console.WriteLine
    ($"The total number of even numbers
    between 1 and 100 is: {result.Count}");
```

Handling Exceptions in PLINQ

Leverage AggregateException to handle exceptions that occur when working with PLINQ using C#. The following code snippet shows how AggregateException can be used to handle exceptions in PLINQ:

```
try
{
    int[] integers =
        Enumerable.Range(1, 100).ToArray();

    var result = integers
        .AsParallel()
        .Select(
            n =>
            {
                if (n % 100 == 0)
                    throw new
                        InvalidOperationException
                        ("Error occurred");
                return n;
            }
        )
        .Where(n => IsPrimeNumber(n))
        .ToList();
}
catch (AggregateException ae)
{
    foreach (var ex in
        ae.InnerExceptions)
    {
        Console.WriteLine
            ($"Exception occurred:
            {ex.Message}");
    }
}
```

```
}
}
```

The IsPrime() method is given below:

```
static bool IsPrime(int integer)
{
    if (integer <= 1)
        return false;
    if (integer == 2)
        return true;

    for (int i = 2;
        i * i <= integer; i++)
        if (integer % i == 0)
            return false;
    return true;
}
```

The degree of parallelism denotes the number of processors or cores used by PLINQ to execute a query. PLINQ allows you to control the degree of parallelism as well. For example, you can programmatically restrict the number of processors used in your PLINQ query, as shown in this code snippet:

```
var results = integers.AsParallel()
    .WithDegreeOfParallelism(2)
    .Select(n => n * 5)
    .ToList();
```

In C#, you can use the ThreadPool class in lieu of creating and destroying threads. In this case, the Task is executed by a thread pertaining to the ThreadPool. The ThreadPool class helps you to queue tasks.

```
ThreadPool.QueueUserWorkItem((state) => {
    Console.WriteLine
        ("This is a thread pool thread.");
});
```

The goal of PLINQ is to optimize the performance of a query while ensuring that the data returned from execution of the query is accurate. Because ordering can be expensive computationally, PLINQ doesn't order the elements of the source sequence. Term ordering describes the ability to arrange the elements of a sequence or collection based on one or more predefined criteria that determines what should be ordered. The result set obtained from execution of PLINQ queries may or may not be ordered.

Use PLINQ when the workload is compute-intensive, the dataset is large, and the system has enough resources (CPU cores). Remember that PLINQ may become a bottleneck for smaller datasets.

Here is an example PLINQ query that doesn't order the data:

```
var data = integers.AsParallel().
    Where(n => n % 2 == 0).ToList();
```

Listing 8: The ThreadSynchronizationExample class

```
public class ThreadSynchronizationExample
{
    private static readonly
    object objLock = new object();
    private static readonly Semaphore
    semaphoreObj = new Semaphore(3, 5);
    private static readonly ReaderWriterLockSlim
    readerWriterLock = new ReaderWriterLockSlim();

    public void AccessSharedDataUsingLock()
    {
        lock (objLock)
        {
            //This is the critical section
        }
    }

    public void AccessSharedDataUsingMonitor()
    {
        Monitor.Enter(objLock);
        try
        {
            //This is the critical section
        }
        finally
        {
            Monitor.Exit(objLock);
        }
    }

    public void AccessSharedDataUsingSemaphore()
    {
        semaphoreObj.WaitOne();

        try
        {
            //This is the critical section
        }
        finally
        {
            semaphoreObj.Release();
        }
    }

    public void AccessSharedDataUsingMutex()
    {
        Mutex mutexObj = new Mutex
        (false, "MyExampleMutex");

        if (mutexObj.WaitOne())
        {
            try
            {
                //This is the critical section
            }
            finally
            {
                mutexObj.ReleaseMutex();
            }
        }
    }

    public void
    AccessSharedDataUsingReaderWriterLock()
    {
        readerWriterLock.EnterReadLock();
        try
        {
            //Read data
        }
        finally
        {
            readerWriterLock.ExitReadLock();
        }
        readerWriterLock.EnterWriteLock();
        try
        {
            //Write data
        }
        finally
        {
            readerWriterLock.ExitWriteLock();
        }
    }
}
```

You can enable ordering of the result set in PLINQ as shown below:

```
var data = integers.AsParallel().
    AsOrdered().Where
    (n => n % 2 == 0).ToList();
```

To improve performance, turn off ordering as shown below:

```
var data = integers.AsParallel.
    Where(n => n > 25).AsUnordered().
    Select(n => n % 2 == 0).ToList();
```

Concurrency

Concurrency is defined as the capability of an operating system to allow several processes or threads to run at the same time. This is one of the most important features in modern operating systems enabling efficient use of CPU resources and enhancing system performance by enabling you to execute multiple tasks simultaneously. Concurrent

execution can happen within a single processor by sharing the CPU time or in computers with multiple processors running concurrently. Concurrency provides for efficient use of resources by performing many operations simultaneously, which leads to faster completion times.

Although you can achieve concurrency in a single processor, parallelism is only possible on multi-core systems or distributed computers. This is because a single processor can execute one and only one thread at a given point of time.

Thread Safety and Synchronization Primitives

Synchronization is an imperative in multi-threaded applications to ensure that your applications execute correctly and maintain consistency of shared resources. There are several synchronization primitives available as part of .NET Core that can help you implement thread safety and synchronization in your applications.

Thread safety guarantees that shared data is accessed and modified by several threads while ensuring that these threads don't corrupt the data or cause unexpected results. Synchronization primitives are language constructs or methods used for ensuring thread safety. A critical section is a shared re-

source that can be accessed by multiple threads where one and only one thread can access it at any given time.

I'll examine some of the synchronization primitives available in .NET Core in this section.

A **lock statement** is a synchronization primitive that enables one thread to enter a critical section of code at a given point in time while all other threads that try to access the shared resource are blocked until the lock on the shared resource is released.

```
private static readonly
object objLock = new object();

lock (objLock)
{
    //This is the critical section
}
```

A read-only object can be initialized only at the time when the object is created or inside the constructor of the class to which it belongs.

You can also implement thread safety using another synchronization primitive, the **Monitor class**, that provides several methods to implement thread synchronization such as Enter, Exit, etc., as shown in the code snippet below:

```
Monitor.Enter(objLock);

try
{
    //This is the critical section
}
finally
{
    Monitor.Exit(objLock);
}
```

You can also use the Monitor.TryEnter method, as shown in the code snippet below:

```
try
{
    if (Monitor.TryEnter(lockObj, 10))
    {
        //This is the critical section
    }
}
catch (Exception ex)
{
}
finally
{
    Monitor.Exit(lockObj);
}
```

The **Semaphore class** can be used as a synchronization primitive to allow multiple threads to access shared resources while restricting the number of threads allowed

to execute the critical section concurrently. To do this, a semaphore count is obtained that's incremented when a thread enters the semaphore and decremented when the thread relinquishes control of the semaphore. When the semaphore count reaches its maximum, all threads are blocked from accessing the shared resource.

```
private static readonly Semaphore
semaphoreObj = new Semaphore(3, 5);

semaphoreObj.WaitOne();

try
{
    //This is the critical section
}
finally
{
    semaphoreObj.Release();
}
```

A mutex is a synchronization primitive used to lock a shared resource exclusively, i.e., it allows you to acquire a mutually exclusive lock where only one thread can have access to the shared resource at given point of time. You can take advantage of the **Mutex class** in C# to create mutexes. You can call the constructor of the Mutex class with the name of mutex as a parameter.

The following code snippet illustrates how you can work with Mutex in C#:

```
Mutex mutexObj = new Mutex
(false, "MyExampleMutex");

if (mutexObj.WaitOne())
{
    try
    {
        //This is the critical section
    }
    finally
    {
        mutexObj.ReleaseMutex();
    }
}
```

The **ReaderWriterLockSlim class** is another synchronization primitive that can be used whenever many threads read from and one thread writes to a shared resource. It achieves this by acquiring a read lock for reading operations and an exclusive write lock for writing operations. It's an optimized version of the ReaderWriterLock class designed to manage access to a resource that's frequently read and occasionally written.

The following piece of code shows how you can work with the ReaderWriterLockSlim class:

```
private static
readonly ReaderWriterLockSlim
readerWriterLock =
new ReaderWriterLockSlim();
```


SPONSORED SIDEBAR

CODE Is Hiring!

CODE Staffing is accepting resumes for various open positions ranging from junior to senior roles. We have **multiple openings** and will consider candidates who seek full-time employment or contracting opportunities.

For more information:
www.codestaffing.com.

```
readerWriterLock.EnterReadLock();
try
{
    //Read data
}
finally
{
    readerWriterLock.ExitReadLock();
}

readerWriterLock.EnterWriteLock();
try
{
    //Write data
}
finally
{
    readerWriterLock.ExitWriteLock();
}
```

The complete source code is given in **Listing 8**.

Points to Ponder

Asynchrony, parallelism, and concurrency, if used judiciously, can boost an application's performance and scalability considerably. Be careful to write your code in such a way that deadlocks and race conditions are avoided. Of course, not all of them, i.e., asynchrony, parallelism, or concurrency, are a good choice in all scenarios. The rule of thumb in determining which of them should be used depends on whether your program is CPU-intensive or I/O-intensive.

Joydip Kanjilal
CODE



Sep/Oct 2024
Volume 25 Issue 6

Group Publisher
Markus Egger

Editor-in-Chief
Rod Paddock

Managing Editor
Ellen Whitney

Content Editor
Melanie Spiller

Writers in This Issue

Markus Egger
Ashleigh Lodge
Jason Murphy

Joydip Kanjilal
Sahil Malik
Paul D. Sheriff

Technical Reviewers

Markus Egger
Rod Paddock

Production

Friedl Raffener Grafik Studio
www.frigraf.it

Graphic Layout

Friedl Raffener Grafik Studio in collaboration
with [onsight \(www.onsightdesign.info\)](http://onsightdesign.info)

Printing

Fry Communications, Inc.
800 West Church Rd.
Mechanicsburg, PA 17055

Advertising Sales

Tammy Ferguson
832-717-4445 ext. 26
tammy@code-magazine.com

Circulation & Distribution

General Circulation: EPS Software Corp.
Newsstand: Ingram Periodicals, Inc.
International Bonded Couriers (IBC)
Media Solutions
Source Interlink International

Subscriptions

Circulation Manager

Colleen Cade
832-717-4445 ext. 28
ccade@codemag.com

US subscriptions are \$29.99 USD for one year. Subscriptions outside the US are \$50.99 USD. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa and Discover credit cards accepted. Back issues are available. For subscription information, email subscriptions@code-magazine.com or contact customer service at 832-717-4445 ext. 9.

Subscribe online at
www.code-magazine.com

CODE Developer Magazine
EPS Software Corporation / Publishing Division
6605 Cypresswood Drive, Ste 425, Spring, Texas 77379 USA
Phone: 832-717-4445



CUSTOM SOFTWARE DEVELOPMENT

STAFFING

TRAINING/MENTORING

SECURITY

**MORE THAN JUST
A MAGAZINE!**

Does your development team lack skills or time to complete all your business-critical software projects? CODE Consulting has top-tier developers available with in-depth experience in .NET, web development, desktop development (WPF), Blazor, Azure, mobile apps, IoT and more.

**CONTACT US TODAY FOR A COMPLIMENTARY ONE HOUR TECH CONSULTATION.
NO STRINGS. NO COMMITMENT. JUST CODE.**

codemag.com/code

832-717-4445 ext. 9 • info@codemag.com



UNLOCK STAFFING EXCELLENCE

Top-Notch IT Talent, Contract Flexibility, Happy Teams, and a Commitment to Customer Success Converge with CODE Staffing

Our IT staffing solutions are engineered to drive your business forward while saving you time and money. Say goodbye to excessive overhead costs and lengthy recruitment efforts. With CODE Staffing, you'll benefit from contract flexibility that caters to both project-based and permanent placements. We optimize your workforce strategy, ensuring a perfect fit for every role and helping you achieve continued operational excellence.

Ready to Discuss Your IT Staffing Needs?

Visit our website to find out more about how we are changing the staffing industry.



Website: codestaffing.com

Yair Alan Griver (yag)

Chief Executive Officer

Direct: +1 425 301 1590

Email: yag@codestaffing.com