Please note: not all items are included in these notes. Please see the videos for details.

Terminology

- In late 2020 Power Apps changed its terminology. For example, entities became tables.
- However, in programming terms, the old terminology remains.
    - In Web API, you use the Odata protocol, which uses data structured defined as EntityTypes.
    - In .NET SDK, there is an Entity class.
- So:
    - Power Apps Tables are programming entities.
    - Columns are attributes in .NET or properties in Web API.
    - Rows are records.
    - Yes/Nos are Booleans
- Plug-ins
    - Code which runs when an event triggers in the Dataverse – real-time
- Custom code components
    - Used to provide data visualization components in canvas apps.
- Azure Functions
    - Can be called from apps (and others), and communicate with other systems.
- Webhooks
    - To make calls using APIs to a web app. Can be synchronous or asynchronous.
- Azure Event Hubs
    - A highly scalable publish/subscribe service, ingesting millions of events per second.
- Azure Service Bus
    - Relays message data between the Dataverse and Azure. Other applications can then get the messages from the Service Bus.
- Power Automate Hub and Logic Apps
    - Creating graphical workflows. Developed for different audiences in mind (see topic 4).
- Web API
    - For communicating with the Dataverse.

Validate requirements and design technical architecture

1. Design and validate the technical architecture for a solution

- Data – where is it to be stored?
    - Is it new data?
        - If so, can it be stored in the Dataverse?
        - You would be able to use business rules, indexes and other features of Dataverse.
    - Can you read/write from an existing system?
        - Can you use a connector to directly read it?
            - Real-time data, able to read and modify existing data.
        - If not, can you use a dataflow (Power Automate flow) to make copies of the existing data for your solution's use?
            - Snapshot data, read-only.
    - Design table structure
        - With data types, relationships and indexes.

        - Star and snowflake schema.
- Power Apps
    - Canvas or model-driven?
        - Do you need pixel-perfect app? If so, choose a canvas app.
        - Are you using a connector to read (at least, some of) the data? If so, choose a canvas app.
        - Can it rely on data solely from the Dataverse? If so, you may wish to choose a model-driven app.
        - Do you want to use less code, business process flows, workflows or actions? Use a model-driven app, if possible.
    - What dashboards, pages and reports are to be included?
    - Buttons for navigating, saving (and not saving) data.
    - Use additional media (photographs, video, audio)?
- Flows
    - Do you need data flowing from one source to another, and maybe transforming?
    - Would it make sense to use Power Automate/Logic Apps,
    - Should you add business logic using client scripting, PCF components, command buttons, or plugins.
- Azure
    - Do you need additional functionality, which could be from Azure? For example, Azure Functions, or registering service endpoints in Azure?

2. Design authentication and authorization strategy

- Do you need any authentication?
    - For example, is it for everyone?
    - However, you might want some protection, otherwise hackers might use it just to be annoying.
- How do you want authentication?
    - Do you need to log into Azure Active Directory?
        - Can this use Multi-factor Authentication (MFA)?
            - This can include FIDO2 Security Key (hardware), Microsoft Authenticator (app), and Text messaging (SMS).
    - Does the user need to have a Power Apps license?
    - Can you use an access key/secret to use (for example) an Azure Function?
- Authorization
    - Do you want to restrict access to the app?
    - Need to give the minimum amount of access.
    - In Azure, you can use Role-Based Access Control to give minimum access to objects.
    - Do you want to restrict access to any of the forms?
    - Do you want to restrict Create, Read, Update and Delete access to a table's rows?
    - Do you want to restrict access to individual columns in a table? (Field-level security)

3. Determine whether you can meet requirements with out-of-the-box functionality

- Canvas app
    - Allows you to use formulas, and data sources other than Dataverse.
    - Allows you to insert buttons and other input and media output components.
- Model-driven apps
    - Low-code, create forms, views, charts, dashboards and site maps for apps relatively swiftly.

    - Create business process flows to guide users.
- Power Automate
    - Connect multiple data sources together, either:
        - manually (button/instant),
        - on a schedule, or
        - automatically, based on a trigger.
- Power BI
    - Create interactive visualizations
    - Insert Power Apps and Power Automate flows
- Power Virtual Agents
    - Chatbots
- Artificial Intelligence
    - Extract information from documents,
    - Extract key elements, relevant words, language and entities (categories) from text, or categorise or classify sentiment from text data
    - Detect objects from images.
- Allow these apps to communicate.
    - Integrate Power BI into apps, Power Automate into Canvas apps, or AI into apps.

4. Determine when to use Logic Apps versus Power Automate flows

- Both Logic Apps and Power Automate allow you to create workflows.
    - They use the same workflow designer and the same connectors.
- The target audience, and therefore the advanced tools, are different:
    - Power Automate is designed for non-developers, such as office workers and business users.
        - You can design flows in-browser and in mobile app.
        - It is more integrated into Power Platform.
    - Logic Apps is designed for IT pros.
        - You can use VS Code and VS with Logic Apps.
        - You can use the additional security in Azure, include Microsoft Defender for Cloud.
        - You can also use Azure DevOps for source control, testing, automation etc.
- The pricing is different:
    - Power Automate is part of the Power Platform, with plans starting at $15 per user per month.
    - On the Consumption plan, the first 4,000 actions in Logic Apps are free.
        - An additional 1,000,000 executions are around $25.
        - 1,000 connector executions cost $1. You also need to pay for Data Retention.
- They can be integrated:
    - For example, a flow can call a logic app.

5. Determine when to use serverless computing, plug-ins, or Power Automate

- Both Azure Functions and Logic Apps use serverless workloads.
    - Azure Functions use serverless compute, with code in C#, Java, JavaScript, PowerShell or Python.
    - Azure Logic Apps use serverless workflows.
        - You can use ready-made actions.
    - "Serverless" means that you don't control the server. Equally, in a Consumption plan, you don't pay for the time you are not using the server (probably, most of the time).

- With Azure Functions, you can:
    - Add an endpoint for your web applications,
    - Run code when file are uploaded/changed in BLOB storage, or documents are created/updated in Cosmos DB,
    - Chain functions together using durable functions.
    - Run code on a schedule,
    - Process message queues using Queue Storage, Service Bus, or Event Hubs,
    - Process data from IoT devices.
- A Plug-in is a .NET assembly (using, for example, C#) which adds a feature to an existing program.
    - You can attach them to Dataverse messages, so that the steps run when (for example) a row is added in a Dataverse table.
        - They can perform well and are powerful.
        - However, they require developer skills.
    - They are written using code.
- Power Automate also runs workflows, and is designed for non-developers, such as office workers and business users.
    - They can also run based on Dataverse triggers.
    - They are created using a GUI. No need to write code.

6. Determine when to build a virtual table data source provider and when to use connectors

- Connectors can provide from a data source:
    - Tables of data, and/or
    - Actions.
- Popular connections include:
    - Microsoft Dataverse,
    - Cloud storage, such as Azure Blob, Box, Dropbox, Google Drive, OneDrive, and OneDrive for Business
    - Dynamics AX,
    - Excel,
    - Microsoft Translator,
    - Office 365 Outlook or Users,
    - Oracle,
    - Power BI,
    - SharePoint,
    - SQL Server and
    - Twitter.
- To add a new connection, go to Dataverse – Connections – "+New connection".
- Why not use connections all the time?
    - Model-driven apps work with tables in the Dataverse only. Connections cannot be used.
- A virtual table is a table defined in Dataverse but not stored there.
    - When required, its state is then (during runtime) retrieved from the Virtual Table Data Provider.
        - This could be an OData v4 provider, which is installed by default in the Dataverse, and uses port 443. It allows for CRUD (created, read – retrieve and retrieve multiple – update and delete operations).
        - It could be Azure Cosmos DB, which you can get from AppSource.

        - It could also be a custom Virtual Table Data Provider.
- Requirements for virtual tables include:
    - It must be Organization-owned, not User-owned.
        - Access can be turned on/off based on a user's security role.
        - Field-level security is not supported.
    - The external data must be able to be modelled as a Dataverse table. For example:
        - It must have an associated GUID primary key. (However, you can also use number primary keys with the new Virtual Connector, which is now in preview).
        - Table properties must be allowed in Dataverse table columns.
        - Table relationships must be modelled in the Dataverse.
        - There can be no calculated or rollup columns.
        - You cannot use the Currency or Image data types.
    - Auditing, chart, dashboards are not supported.
    - Searching is not supported, as data is not persisted.
    - It cannot represent an activity.
    - You cannot use business process flows on this table.
    - Virtual tables cannot be changed to standard tables (and vice versa).
    - To create a Virtual Entity Data Source, go to:
        - Settings – Advanced Settings – Administration – Virtual Entity Data Source.
        - Go to New, choose the correct Virtual Table Data Provider, and click OK.
        - Provide the relevant details.
    - To add a table:
        - Go to the Power Apps portal – Solutions, and click on "Switch to classic".
        - Select the relevant solution.
        - Go to Entities (left-hand side), then New.
        - Select "Virtual entity", and enter the relevant information.
        - Then create the columns.
        - Then add columns to the main form.
- There is a third way – Virtual connectors in Dataverse solution.
- In the public preview, you can create virtual tables using:
    - SQL Server,
    - Microsoft Excel Online (Business), and
    - Microsoft SharePoint.
- You can download it in the AppSource. However, limitations included:
    - Queries are limited to return 250 records.
    - Text entries cannot exceed 4,000 characters.
    - Audits and rollups are not allowed.

Design solution components

7. Design a Microsoft Dataverse data model

- Covered in PL-100/200 exams.

8. Design Power Apps reusable components

- Canvas Power App Components - Covered in PL-100/200 exams.
- PCF Controls – covered in topics 51-56.

9. Design custom connectors

- Covered topics 69-72.

10. Design server-side components

- Business Rules – covered in topic 45 (and PL-100/200 exam).
- Plug-ins – covered in topics 60-68.
- Business Process Flows – covered in topic 44 (and PL-100/200 exam).
- Real-Time Classic Workflows – covered in PL-100/200 exam.

11. Determine when to extend business process flows by using server-side and client-side code or Power Automate

- Power Automate
    - Designed using a GUI (developer optional).
    - Used it to (for example):
        - Automate business processes,
        - Send email reminders when tasks are overdue,
        - Move business data between different data sources.
- Client-side code
    - Uses JavaScript/Typescript API methods (so needs a developer).
    - Run following events, such as:
        - A model-driven form loads,
        - Data is changed in a column,
        - Data is changed in an item within a model-driven form, or
        - Data is saved in a model-driven form.
    - Results can include:
        - Changing the active business process flow,
        - Move the active stage to a different stage,
        - Hide/show form tabs and columns,
        - Show messages (for example, Xrm.Navigation.openAlertDialog)
        - Instantly updating fields in form, and
        - Fetching related data
- Server-side code
    - Results can include:
        - Send email to users,
        - Create activity(s) based on the active stage,
        - Close a business process.
- See also topic 46.

Describe Microsoft Power Platform extensibility points

12. Describe Power Virtual Agents extensibility points including Bot Framework skills and Power Automate flows

- Covered in PL-200 under the topic of "Fallback topics".

13. Describe Power Apps portal extensibility points including CRUD APIs and custom styling

- Custom styling has been covered in PL-200 exam.
- Power Apps portal extensibility points are covered in topics 60-68.

14. Describe Dataverse custom APIs and their uses

- Allows you to extend functionality of the Power Platform using logic which is started with a plug-in.
    - They are an alternative of Custom process actions, which provides a no-code way to implement this logic. However, Custom process actions has various limitations.
- Uses: With the Dataverse Web API, you can have both actions and functions.
    - Actions make changes to the data.
    - Functions retrieves data.
        - You can use a GET request to retrieve the data.
    - Custom Process Actions only allow for actions.
- Advantages include:
    - With Custom API, you can say that a user must have a particular privilege to call the message – or it will return an error.
        - This is not possible in Custom Process Actions.
    - With Custom API, the message can be marked as private.
        - This indicates that you do not support others with this message.
            - It stops it from appearing in the $metadata service document,
            - However, others can use your message if they know about it.
        - You can set the "Is Private" or "IsPrivate" property before creating a managed solution.
        - With Custom Process Actions, it is publicly exposed – other people can use it.
    - With Custom API, you can bind an operation to a table collection.
        - This restricts the operation's context to that type.
    - You can modify a Custom API by editing a solution.
    - Custom APIs are limited to 2 minutes.
        - Custom Process Actions are limited to 2 minutes per part.

Configure security to support development

15. Troubleshoot operational security issues

- Users need to have the right degree of security.
- To look at a user's permissions:
    - Go to the Power Platform admin center – Environments – [select an environment].
    - Then go to Settings – "Users + permissions" – Users.
    - Select a user
    - Click on "Run diagnostics".
    - Review the details.
- Issues may include:
    - User has no roles.
        - Add security roles to a user.
    - User doesn't have sufficient permissions
        - In – "Users + permissions" – Users, click on the user and:
            - go to … - Manage Roles, note the security role, and then close this dialog box.
            - Go to Security – Security Roles and select the relevant security role.
            - Select the "Core Records" tab and ensure that "User Entity UI Settings" is at least "User".
    - User doesn't have a license / user doesn't belong to the organization

        - Verify that there is a license in Microsoft 365 – Billing – Licenses.
    - User isn't a member of the environment's security group
        - The system administrator should verify that the environment is associated with an AAD group.
            - Go to Power Platform admin center – Environments – [select an environment], then click "Edit" and look at the "Security group".
        - Make sure that the relevant user is associated with that group.
            - It may take time for it to be synchronised.
- Other considerations?
    - Do users need access to all of a table's columns/fields? Maybe use field level security.
    - Do they have access to the right form in a model-driven app?

16. Create or update security roles and column-level security profiles

- Covered in PL-100/200 exams.

17. Configure business units and teams

- Covered in PL-100/200 exams.

Implement tables and columns

18. Configure tables and table options

- Covered in PL-100/200 exams.

19. Configure columns

- Covered in PL-100/200 exams.

20. Configure relationships and types of behaviors

- Covered in PL-100/200 exams.

Implement application lifecycle management (ALM)

21. Create solutions and manage solution components

- Covered in PL-100/200 exams.

22. Import and export solutions

- Covered in PL-100/200 exams.

23. Manage solution dependencies

- Covered in PL-100/200 exams.

24. Implement source control for projects including solutions and code assets

- Exported solutions cannot be added usefully to source control systems.
    - They are not structured to allow source code management to evaluate differences and capture changes.
- However, you can use the SolutionPackager tool to export (unpack) a solution from.zip to a folder containing multiple XML files (or vice versa), and then add this to your source control system (such as GitHub).
    - The SolutionPackager tool is included in the Microsoft.CrmSdk.CoreTools NuGet package.
    - You can also use GitHub actions.
- SolutionPackager is a command-line tool, which requires the following parameters for the solution .zip file:

    - /action – either:
        - /action: Extract – extracts the .zip to a folder,
        - /action: Pack – packs a folder into a .zip file.
    - /zipfile: <file path> - the path and name of the .zip file.
    - /folder: <folder path> - the path of the folder used for extracting or packing.
- The following parameters can be used:
    - /packagetype: - either Unmanaged, Managed or Both.
        - Not usually needed, as the package type can be read from the .zip file. The default is Unmanaged.
    - /errorlevel: <level>
        - The level of logging information to output. The default is Info. Other values are Off, Error, Warning and Verbose.
    - /log: <file path>
        - Logging information will be created in/appended to this file.
    - /localize or /loc
        - Extract or merge string resources into .resx files.
- The following parameters can be used when extracting:
    - /allowWrite: Yes or No.
        - The default is Yes. If No, then no files will be written or deleted.
    - /allowDelete: Yes, No or Prompt.
        - The default is Prompt.
            - If Yes, any existing files in the target folder will be deleted.
            - If No, they will not be deleted.
            - If Prompt, the user will be prompt to allow or deny all delete operations.
    - /clobber – if this is used, read-only files will be overwritten/deleted.

25. Create and use environment variables

- Covered in PL-100/200 exams.

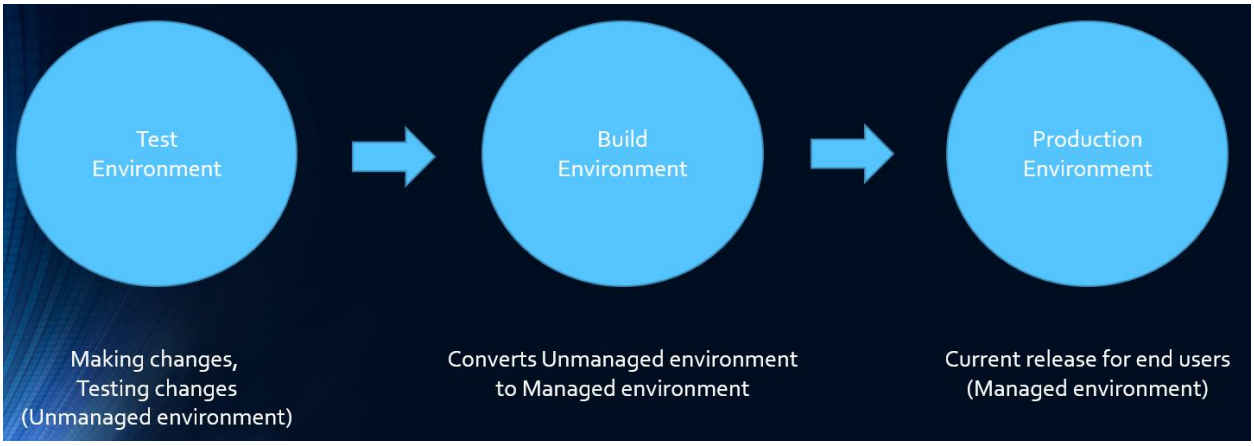
26. Describe how to use Package Deployer and associated tools to create a package

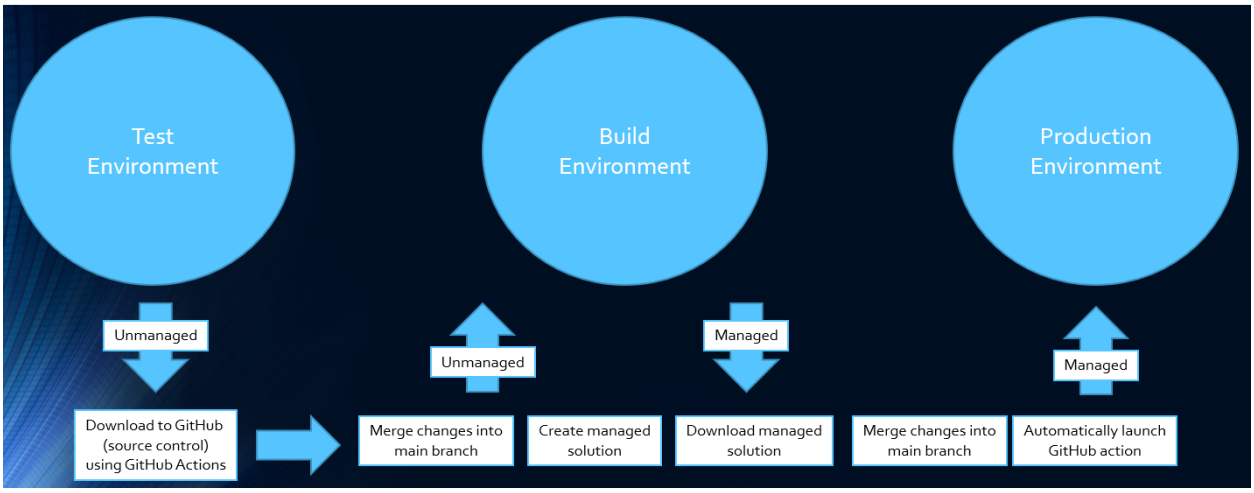
- Package Deployer, which uses Visual Studio 2015 or later, can create packages. They can include:
    - Dataverse Solution file(s),
    - Flat files,
    - test data files that have been exported using the Configuration Migration tool.
    - Custom code to be run while or following the package deployment.
    - HTML content to be shown at the start and finish of the deployment.
- To create the packages, you need:
    - the Microsoft .NET Framework 4.6.2,
    - If you are using Visual Studio 2015, the NuGet Package Manager
    - The Microsoft Dynamics CRM SDK templates.
- In Visual Studio, create a project:
    - Go to New Project
    - Find and select the CRM Package.
    - Give the project a name and location.
- Add files to the project
    - In Solution Explorer, right-hand click the PkgFolder folder, click Add – Existing File and add solutions and files.

        - For each file, go to Properties and ensure that "Copy to Output Directory" to "Copy Always".
- Update the HTML files (optional).
    - In Solution Explorer, go to PkgFolder – Content – en-us.
    - The two folders WelcomeHTML and EndHTML display information at the start and end of the process.
    - You can also duplicate the en-us folder to (for example) fr-FR, and modify these HTML files.
- Add configuration values:
    - Go to PkgFolder and double-click the ImportConfig.xml
    - The parameters are:
        - installsampledata – true or false
        - waitforsampledatatoinstall – true or false
        - crmmigdataimportfile – default file name for test data exported using the Configuration Migration tool (not needed for the PL-400 exam).
        - solutions – the individual solutions contains:
            - solutionpackagefilename – the .zip name for your solution
            - overwriteunmanagedcustomizations – true or false (optional). If omitted, unmanaged customizations in the existing solution are maintained.
            - publishworkflowsandactivateplugins – true or false (optional). If omitted, after the solution is imported, workflows are published and plug-ins are activated.
        - filestoimport – contains configimportfile and zipimportdetails notes.
        - configimportfile contains:
            - filename. If a zip file, you need a zipimportdetails node.
            - filetype – csv, xml or zip.
            - associatedmap – the Dataverse import data map. If omitted, it uses the system determined import data map name.
            - importtoentity – name of the exe in the zip file, URL or .msi file.
            - datadelimiter – singlequote or doublequote.
            - fielddelimiter – comma, colon or singlequote.
            - isfirstrowheader – true or false.
            - isrecordownerateam – true or false.
            - owneruser – the user ID.
            - waitforimporttocomplete – true or false.
        - zipimportdetails contains 1 or more zipimportdetail nodes.
        - The zipimportdetail node contains:
            - filename,
            - filetype – csv or xml, and
            - importtoentity – name of the exe in the zip file, URL or .msi file.
        - filesmapstoimport, which contains configimportmapfile, which contains the filename of a map file.
    - Click "Save All".
- Add custom code (optional):
    - Double-click on the PackageTemplate.cs
    - Add any custom code.
- Build the package.

- To deploy the packages, you need to download the Package Deployer tool from NuGet (Microsoft.CrmSdk.XrmTooling.PackageDeployment.WPF).
    - Go to where you have built the package.
        - It is in your Visual Studio project folder, then Bin\Debug.
    - Copy:
        - The PackageName folder. Because you have ensure that "Copy to Output Directory" to "Copy Always", this contains your solutions, import configuration and contents.
        - The PackageName.dll. This contains your code.
    - Go to your Package Deployer tools folder
        - It will be in a folder called "tools". It contains the PackageDeployer.exe file.
    - Paste the PackageName.dll and folder into that folder.
- To use the Package Deployer tool GUI:
    - Double-click on the PackageDeployer.exe file. This runs the Package Deployer:
        - Click continue, then enter your Dataverse details.
            - You can also check "Always display list of available orgs", if you wish.
        - Select the package to be deployed, and follow the rest of the instructions.
- To use the Package Deployer at the command line:
    - Enter in PowerShell, at the appropriate folder, something like:
        - packagedeployer.exe /Settings:"SkipChecks=false|lcid=1033"
        - "SkipChecks" = true will bypass some safety checks, which may speed up the import. However, only use when the target environment does not contain any other solutions.
        - LCID is the locale ID – 1033 is en-US. It is optional – the default language will be used if it is not specified.

27. Application Lifecycle Management (ALM)





Create model-driven apps

28. Configure a model-driven app

- Covered in PL-100/200 exams.

29. Configure forms

- Covered in PL-100/200 exams.

30. Configure views

- Covered in PL-100/200 exams.

31. Configure commands and buttons

- Covered in PL-100/200 exams.

Create canvas apps

32. Create and configure a canvas app or custom page

- Covered in PL-100/200 exams.

33. Implement complex formulas to manage control events and properties

- Covered in PL-100/200 exams.

34. Build reusable component libraries

- Covered in PL-100/200 exams.

35. Test an app by using Test Studio

- In the left-hand pane, go to Advanced (at the bottom) – Tests.
    - Enable “Formula-level error management” for best results.
- This opens the Test Studio.
    - Test steps are a series of instructions or actions.
    - They are put together into a sequence in a “test case”.
    - They can be grouped together in “test suites”.
    - You can validate whether the test has run correctly using “test assertions”, a true/false expression. true means that it has run correctly; false means your test has failed.

- Click “Record” and do your test.
- Then review the steps that have happened.
    - Maybe alter one of the steps as a test.
- Then you can replay it.
- You can also click on the test, and select “Copy play link” to test it externally.
    - Different users can run the test on different machines.
    - You can then use the OnTestCaseComplete expression and the new variable TestCaseResult, with the properties:
        - StartTime, EndTime, TestCase- or TestSuite- Id, -Name, -Description (TestSuite as a series of partial test cases – if Case 1 ends on Screen2, Case2 will start there).
        - Success (true/false), TestFailureMessage (which step failed and why) and Traces (Assert and Trace entries).
- For maximum benefits from Test Studio:
    - Determine which test cases should be automated.
        - Repetitive tests.
        - High business impact functionality tests.
        - Features that are stable and not undergoing significant change.
        - Features that require multiple data sets.
        - Manual testing that takes significant time and effort.
    - Keep test cases small.
        - Makes it easier to isolate test failures.
    - Keep expressions to a single test action.
    - Every test case should have an expected result.
    - Use test suites.
        - For maintenance, group or categorize similar test cases together and describe the purpose and expected results of your test.
- Code used:

```
Set(NumberOfRecordsBefore,CountIf(Accounts,true))

Trace(NumberOfRecordsBefore & "");Notify(NumberOfRecordsBefore & "")

Set(NumberOfRecordsAfter,CountIf(Accounts,true))

Trace(NumberOfRecordsAfter & "");Notify(NumberOfRecordsAfter & "")

Assert(NumberOfRecordsAfter = NumberOfRecordsBefore+1,"Different
number of rows than expected!")

Notify(TestCaseResult.Traces) -> Notify(JSON(TestCaseResult.Traces))
```

36. Embed an app in Microsoft Teams

- Covered in PL-100/200 exams.

Manage and troubleshoot apps

37. Troubleshoot app issues by using Monitor and other browser-based debugging tools

- You can use Monitor on model-driven apps.
    - To launch Monitor, click on the … in Advanced Tools, or next to the app in the Power Apps Portal.
    - Then you should use your model-driven app
    - It checks network activity.

        - Details such as Data in JSON.
        - Formula – which formula caused this event.
        - Request – actual network request made.
        - Response – what was the response, either as a Table or in JSON format.
    - Also:
        - You can also run Monitor by:
            - playing the model-driven app,
            - adding &monitor=true at the end,
            - select "Monitor" on the command bar.
- Additional debugging tools for model-driven apps
    - You can also add parameters to the end of the URL. Start with "&flags=", and then add
        - DisableFormCommandbar = true – removes the command bar from form pages only.
        - DisableFormHandlers – you can see the form handlers by clicking on a Category = FormChecker and Operation = FormEvents.
            - Adding "=true" disables business rules and OnLoad, OnSave, OnChange and TabStateChange event handlers
            - Adding "=" and the event name disabling a specific name – for example "DisableFormHandlers=onload". or "=businessrule"
            - Adding "_" and index disables that index. "true_0" disables all event handlers at index 0.
            - Adding "_", index, "_" and index2 disables the relevant event handlers between index and index2. For example, DisableFormHandlers=onload_1_3 disables OnLoad event handlers of indexes 1, 2 and 3 .
        - DisableFormLibraries disables
            - "=true" – all form libraries (JavaScript web resources) from being loaded
            - "=" and index – form libraries at that index, and
            - "=", index, "_" and index2 – form libraries between index and index2.
        - DisableWebResouceControls = true – disables all web resource controls
        - DisableFormControl = true or = controlname – disables a form control
        - DisableBusinessProcessFlow = true – disables all BPFs.
        - navbar=off (do not use flags= for this one).
        - You can have multiple flags by separating them with a comma (do not include &flags= for the second and subsequent flags).
- You can also use App checker in canvas apps and Solution checker in model-driven apps.
- You can also use your browser tools:
    - Ctrl+Shift+I in Chrome, F12 in Edge/IE,
    - Fiddler (a third-party tool)

38. Identify and resolve connector and API errors

- You can identify service connections with server error by going to Admin center – Analytics – Power Apps – Service Performance, and looking at the "Service connections with server error (HTTP 500)"
    - HTTP Error 500 represents that something is wrong, such as an invalid expression.
    - Alternatively, the custom connector might be incorrect – maybe you've added one too many /s.
        - Check the path and host fields are correctly defined.
- To trap any errors that are happening:
    - In Power Apps, go to Settings – Upcoming features – Preview – and turn On Formula-level error management.
    - You can use:
        - IfError(FirstExpression, SecondExpression).
            - If FirstExpression results in an error, then it will then evaluate SecondExpression.
        - Within SecondExpression, you can use FirstError, which evaluates to the first error found.
            - It is a record.
            - AllErrors evaluates to all errors found. It is a table.
            - Properties include:
                - Kind – category (Number),
                - Message – error message,
                - Source – where the error originated,

            - Observed – where error surfaced to user, and
            - Details.HttpResponse and Details.HttpStatusCode – record.
    - You can also call a Power Automate flow (see topic 43 "Implement error handling")

39. Optimize app performance including pre-loading data and query delegation

- Smaller apps may have better performance.
    - Very big apps, or apps with too many big images can cause issues with performance and load times.
        - Can you break down a very big app into smaller apps?
    - Are you loading graphics, images and video? Can they be optimised for the size of screen?
- Are the network bandwidth and latency issues?
    - Data sources are a big bottleneck, as is refreshing data, due to network traffic.
    - Don't refresh when you don't need to. For example, the Gallery automatically refreshes.
    - Store data in the fastest place possible.
    - Are you loading too much data?
        - Can you load data using the OnVisible event into a collection?
            - Use ClearCollect to load the data – but only up to the "500" row range.
            - Use Concurrent to do the loading in parallel. Instead of ClearCollect(); ClearCollect(), use Concurrent(ClearCollect(), ClearCollect())
    - Can you load a screen's data when needed? See File – Settings – Preview – Delayed load.
        - Are there other preview (beta) or experimental (alpha – may change, break, disappear) features that could be used?
    - Test using WiFi, cell data etc.
- Are there too many controls and screens?
    - Can you use a Gallery instead of Canvas/Data Cards?
- App.OnStart, [Screen].OnVisible
    - App.OnStart runs once, before the app can open (blocking).
    - [Screen].OnVisible runs every time the screen is opened, but can run in the background (non-blocking).
    - App.StartScreen allows you to enter the start screen.
        - This will work separately from App.OnStart, and does not need any Navigate – just the name of the screen – and you can use formulas.
- How long does something take to work? Use a Timer to get the answer in milliseconds?
    - Insert a timer, and Start=varTimerStart
    - Reset(Timer1);Set(varTimerStart, true);…;Set(varTimerStart, false)
- Is the app doing too many functions on the client with a big data set?
    - If so, is it having to load as much information the information first?
        - The default limit is 500. It can be adjusted up to 2,000 by going to File – Settings – Advanced settings – Data row limit for non-delegable queries.
            - However, this will increase the amount of time for the data to be processed.
        - Anything beyond this limit will not be returned – it's as if the dataset was smaller.
    - Not a problem if it is a small dataset.
    - Can you delegate functions onto the source? This requires specific tabular data sources, such as:

    - Dataverse, SharePoint, SQL Server.
- Are you using delegable functions?
    - The following can be delegated.
        - Filtering: Filter, Search and Lookup.
            - However, too many lookups can be time consuming, especially if it is done for multiple rows.
        - Sorting: Sort (a single column, no operators or functions) and SortByColumns.
        - Aggregate: Sum, Average, Min and Max.
        - Table shaping functions partially support delegation: Add-, Drop-, Rename- and Show- Columns.
    - Functions which evaluate to a constant value, such as Today() and Language() don't affect delegation.
    - The following cannot be delegated:
        - A yellow triangle which will show for a non-delegatable function.
        - If, *, /, Mod, Concatenate, &, ExactIn,
        - String manipulation functions such as: Lower, Upper, Left, Mid, Len
        - Signals such as: Location, Acceleration, Compass
        - Other aggregation functions such as: CountRows, CountA, Count, StdevP, VarP
        - Rand and other volatile functions, and collections (Collect, ClearCollect).
        - First, FirstN, Last, LastN, Choices, CountIf, RemoveIf, UpdateIf, GroupBy and Ungroup.
        - If you are using the "Upcoming feature" "Enhanced delegation for Microsoft Dataverse", then CountRows, CountIf, AsType and "in" will be delegated.

Category	Functions	Delegatable?
Filtering	Filter, Search and Lookup	Yes
Sorting	Sort and SortByColumns	Yes
Aggregate (some)	Sum, Average, Min and Max	Yes
Table shaping functions	Add-, Drop-, Rename- and Show- Columns	Partially
Standard	If, *, /, Mod, Concatenate, &, ExactIn	No
String manipulation	Lower, Upper, Left, Mid, Len	No
Signals	Location, Acceleration, Compass	No
Aggregate (some)	CountA, Count, StdevP, VarP	No
Volatile and collections	Rand, Collect, ClearCollect	No
Other functions	First, FirstN, Last, LastN, Choices, RemoveIf, UpdateIf, GroupBy and Ungroup	No
Enhanced delegation	CountRows, CountIf, AsType, in	Maybe

- Have you done improvements, but not republished the app?
- Do you have any plug-ins?
- Have a look at the Dataverse analytics to see if there are any potential performance problem areas:
    - Go to https://admin.powerplatform.microsoft.com/ (Power Platform Admin Center), and go to Analytics – Dataverse – Entity Usage.
- Power Apps Analytics can also provide detailed statistics:
    - Go to https://admin.powerplatform.microsoft.com/ (Power Platform Admin Center), and go to Analytics – Power Apps.
    - Information about overall app usage, location statistics, errors send by end users can help diagnose problem areas.

Configure Power Automate

40. Build a cloud flow

- Covered in PL-100/200 exams.

41. Configure steps to use Dataverse connector actions and triggers

- Covered in PL-100/200 exams.

42. Implement complex expressions in flow steps

- Covered in PL-100/200 exams.

43. Implement error handling

- When you get user input, you need to make sure that it is valid.
    - Sometimes the only way to do this is to do an action, and see if it fails. For example, trying to retrieve details of an employee that doesn't exist.
- To implement error handling in a Power Automate flow:
    - Right-hand click on an action which requires error handling.
    - Go to "Add a parallel branch" – "Add an action".
    - Click the "…" in the top-right of this new action, and select "Configure run after" - and either (or both) "has failed" or "has timed out".
    - Then add an action if it there is a problem.
        - Maybe send an email, add the information to a log, or respond to the user or the service which called the log saying "I cannot find this employee".
- If you have too many calls:
    - You may get one of the following error messages:
        - 408 "Request timeout" error message,
        - 429 "Too many requests",
        - anything starting with 5, showing an internal error.
    - Power Automate automatically handles throttling, including retrying automatically up to twice (used to be four times).
    - To view the policy, click on the … in the top-right of an action, and go to Settings.
    - You can then go to the "Retry policy".
    - You can set up a:
        - Fixed interval policy – retrieves up to X times (default 2) and specify an interval in the ISO 8601 format. It always starts with P (for "period").
            - For days above, it has letters followed by a number:
                - Y(ear), M(onth), W(eek), D(ay),

- For hours or below, it has the letter PT (Period – Time) followed by numbers followed by a letter:
    - H(our), M(inutes), S(econd).
- For example, PT10S means retry every 10 seconds.

- Exponential interval retry policy – adds additional time between each retry. It is random between the minimum and maximum intervals.

HTTP Response Status Code	Error Message
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
405	Method Not Allowed
407	Proxy Authentication Required
408	Request Timeout
411	Length Required
413	Payload Too Large
414	URI Too Long
415	Unsupported Media Type
416	Range Not Satisfiable
417	Expectation Failed
429	Too Many Requests
431	Request Header Fields Too Large
451	Unavailable For Legal Reasons

HTTP Response Status Code	Error Message
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
511	Network Authentication Required

Implement processes

44. Create and configure business process flows

- Covered in PL-100/200 exams.

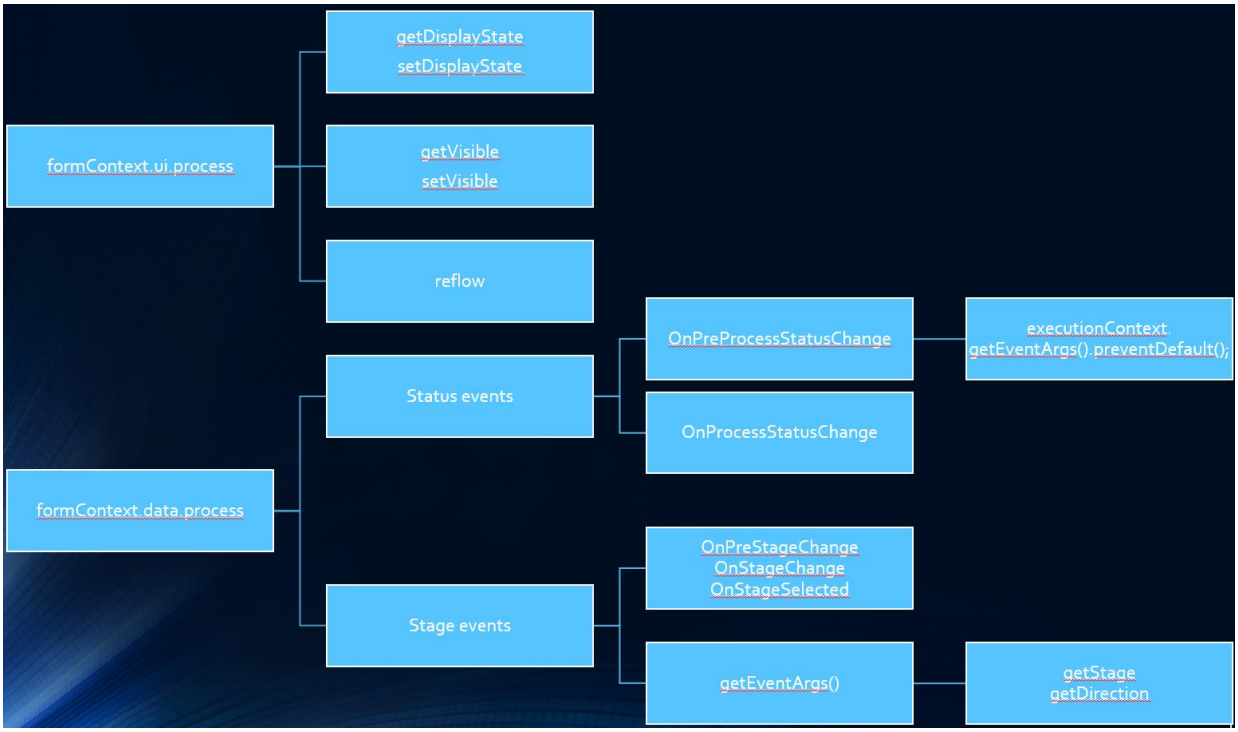
45. Create and configure business rules

- Covered in PL-100/200 exams.

46. Create, manage, and interact with business process flows by using server-side and client-side code

- While creating a business process flow, you can use the following server-side code:
  - Workflow
    - Workflows can be triggered:
      - At the Stage Entry (apart from the first stage), or
      - At the Stage Exit (apart from the last stage).
    - Synchronous classic workflows were covered in PL-200.
  - Action Step
    - Classic workflows can be added into the start, middle or end of a stage.
    - You need to have "As a Business Process Flow action step" checked.
    - Synchronous classic workflows were covered in PL-200.
  - Flow Step
    - Flow steps can be added into the start, middle or end of a stage.
    - Power Automate flows were covered in PL-100.
- The sample code is from https://github.com/microsoft/Dynamics365-Apps-Samples/tree/master/samples-from-msdn/WorkWithBPF
  - It uses the MIT license that allows you to use it without restriction except for an AS IS notice.
- For tables to be able to use business process flows (BPFs):
  - They need the IsBusinessProcessEnabled property = true.
    - Once set to true, it cannot be reset to false.
    - You can also do this in the Entity view (Advanced Settings – Customization – Customize the System – Entity – [Name of table] – check the "Business process flows" checkbox.
- Notes:
  - A BPF row is, by default, created as a Draft.
  - The BPF needs to be activated before you can use it.
    - A new table is created when a BPF is activated – typically using the name of the BPF.
      - For example, a business process flow called BPF may generate a table called new_bpf.
      - It that table is not available, then a table with a GUID extension will be created instead.
  - BPF definitions are stored in the workflow table.
    - LogicalName = workflow, Display name = Process.
  - Stage information can be accessed using Microsoft.Xrm.Sdk.Entity.ProcessStage
  - You can access rows in a BPF table as you would in any other tables, using create, retrieve, update and delete (CRUD) or retrieve.
    - You can change the state of a process instance by changing:
      - statecode to 0 (Active) or 1 (Inactive), and
      - statuscode to 1 (Active), 2 (Finished) or 3 (Aborted)
      - Processes can be aborted at any stage.
      - An active process would have statecode = 0 and statuscode = 1.
      - A process can only be finished when it has reached its last stage of the process.

- If you are adding rows to a table where there is a BPF, then do you want the BPF to be activated?
    - If not, then when creating the new table row, set ProcessID = Guid.Empty.
    - Or you could set ProcessID to a different BPF table, using the table reference. If this happens, this BPF will be used.
    - Otherwise, the default BPF will be used.
        - 46. Create, manage, and interact with business process flows by using server-side and client-side code
- The following Client API events are available for BPF visibility:
    - In formContext.ui.process:
        - getDisplayState and setDisplayState
            - gets/sets "expanded", "collapsed" or "floating"
        - getVisible and setVisible
            - gets/sets true or false to show or hide it
        - reflow
            - refreshes the BPF user interface. You can also specify the parent/next stage.
- The following Client API events are available for interaction with the processes:
    - In formContext.data.process:
        - Status events
            - OnPreProcessStatusChange
                - Occurs before the status changes. Can use executionContext.getEventArgs().preventDefault(); to cancel the process.
            - OnProcessStatusChange
                - Occurs when the status changes.
        - Stage events.
            - OnPreStageChange
                - Occurs before the BPF stage changes.
            - OnStageChange
                - Occurs when the BPF stage changes.
            - OnStageSelected
                - Occurs when a stage is selected.
        - Stage events occur when:
            - the user clicks on Next stage, Move to previous stage, Set Active Stage, or
            - formContext.data.process.moveNext, .movePrevious and .setActivestage methods are used.
        - Stage events allow for a getEventArgs() method to retrieve:
            - getStage, and
            - getDirection (for the OnPreStageChange and OnStageChange events)
    - You can add/remove event handlers by prefixing add or remove to the event name, and putting the function in brackets.



- You can use these methods:
  - Navigation methods:
    - moveNext and movePrevious – for example, formContext.moveNext();
  - Process methods:
    - getId, getName, getStages, isRendered
  - ProcessInstance methods:
    - getProcessInstances and setActiveProcessInstance
  - Instance method:
    - getInstanceId, getInstanceName, getStatus, setStatus
  - Active Stage and ActiveProcess methods:
    - getActiveStage and setActiveStage, getActiveProcess and setActiveProcess
  - Stage methods:
    - getCategory, getEntityName, getID (of the stage), getName (of the stage), getStatus and getSteps.
  - Step methods:
    - getAttribute (column), getName, getProgress, setProgress and isRequired
  - Other methods:
    - getActivePath, getEnabledProcesses (the enabled processes for a table), and getSelectedStage
- Developer Tools
  - You can go to Chrome – More Tools – Developer Tools, and enter:
    - var process = Xrm.Page.data.process;
    - process.moveNext();
    - process.movePrevious();
  - Note – you can dock the Developer Tools by going to the ... and changing "Dock side".
    - Create, manage, and interact with business process flows by using server-side and client-side code

Category	Method name
Navigation	moveNext and movePrevious
Process	getId, getName, getStages, isRendered
Process Instance	getProcessInstances and setActiveProcessInstance
Instance	getInstanceId, getInstanceName, getStatus, setStatus
Active Stage and Active Process	getActiveStage and setActiveStage, getActiveProcess and setActiveProcess
Stage	getCategory, getEntityName, getID, getName, getStatus and getSteps.
Step	getAttribute, getName, getProgress, setProgress and isRequired
Other	getActivePath, getEnabledProcesses, and getSelectedStage

47. Troubleshoot processes

- Part of topics 40-46.

Apply business logic using client scripting

48. Create JavaScript or Typescript code that targets the Client API object model

- JavaScript can be used in:
    - Form Script event handlers
    - Command bar (ribbon) commands (topics 58-60).
        - You can use RibbonDiffXml to add, remove or hide ribbon elements.
    - Web resources and IFRAMEs.
- Note:
    - JavaScript is Mozilla's version,
    - JScript is Microsoft's version,
    - TypeScript
- This avoids accessing the Document Object Model (DOM) of model-driven app pages.
    - Using jQuery is not recommended.
- You can create your JavaScript in Notepad++, Visual Studio Code, Microsoft Visual Studio.
    - or any program which specifically enables creating of JavaScript files.
    - There is no in-built JavaScript editor in Power Apps.
- If you can use Business Rules ,then you should use Business Rules instead of JavaScript.
- The Client API object model uses:
    - executionContext.formContext – the form or item where the code runs.
    - executionContext.gridContext – the grid or subgrid where the code runs.
    - Xrm – used for operations that do not directly affect the data and UI, such as navigating forms, and creating and managing records.
        - You may see Xrm.Page on the web being used instead of executionContext.formContext. Xrm.Page is deprecated (it may be removed in the future), and you should not use.
- executionContext.formContext contains:
    - data object – which contains:
        - attributes – non-table data and controls.
        - entity – contains:
            - entity.attributes for the current record, but only the columns shown on the form.

            - entity.attributes.controls also provide access to controls in the column.
        - save method.
    - process – business process rules process.stages and process.steps on a form.
    - methods:
        - addOnLoad/removeOnLoad – runs/removes a function when form data is loaded.
        - getIsDirty – whether the data has been modified.
        - isValid – whether there are any required columns which are empty
        - refresh – refreshes and optionally saves the form data
        - save – saved the form data
- ui object – which contains:
    - formSelector.items – query forms available for the current user.
    - navigation.items – options in the navigation bar
    - controls – access to the controls on the form
    - process – business process rules stages and steps on a form.
    - quickForms – access to the quick view Form controls.
    - tabs and tabs.sections – access to Form tabs
    - methods:
        - addOnLoad and removeOnLoad
        - close – closes the form
        - getFormType – returns:
            - 0 = Undefined,
            - 1 = Create (Quick Create forms always return 1),
            - 2 = Update,
            - 3 = Read Only,
            - 4 = Disabled,
            - 6 = Bulk Edit.
        - getViewPortHeight and getViewPortWidth – returns the height and width of the form body in pixels.
        - setFormEntityName – sets the table name to be displayed
        - setFormNotification(message, level, uniqueID) – adds a notification
            - message = message text,
            - level = ERROR, WARNING and INFO.
            - uniqueID – string
        - clearFormNotification(uniqueID) – removes the notification
- getControl – this is a shortcut to formContext.ui.controls.get
    - For example, formContext.getControl("nameofcontrol")
- If it is part of a composite control, you will need to give both the composite and the column name, separated by "_compositionLinkControl_".
- From here, you can add
    - .addNotification(). In ({ and }) you would add
        - actions (Optional). This contains an array of messages and actions.
        - messages – an array of strings, but currently only the first will be displayed in bold text. Should be up to 50 characters long.
        - notificationlevel
            - RECOMMENDATION (which allows for an Apply and Dismiss button, and an "i" or a lightbulb next to the control) or

        - ERROR (only the Dismiss button, and a red "X" next to the control).
    - uniqueId – string.
    - For example:

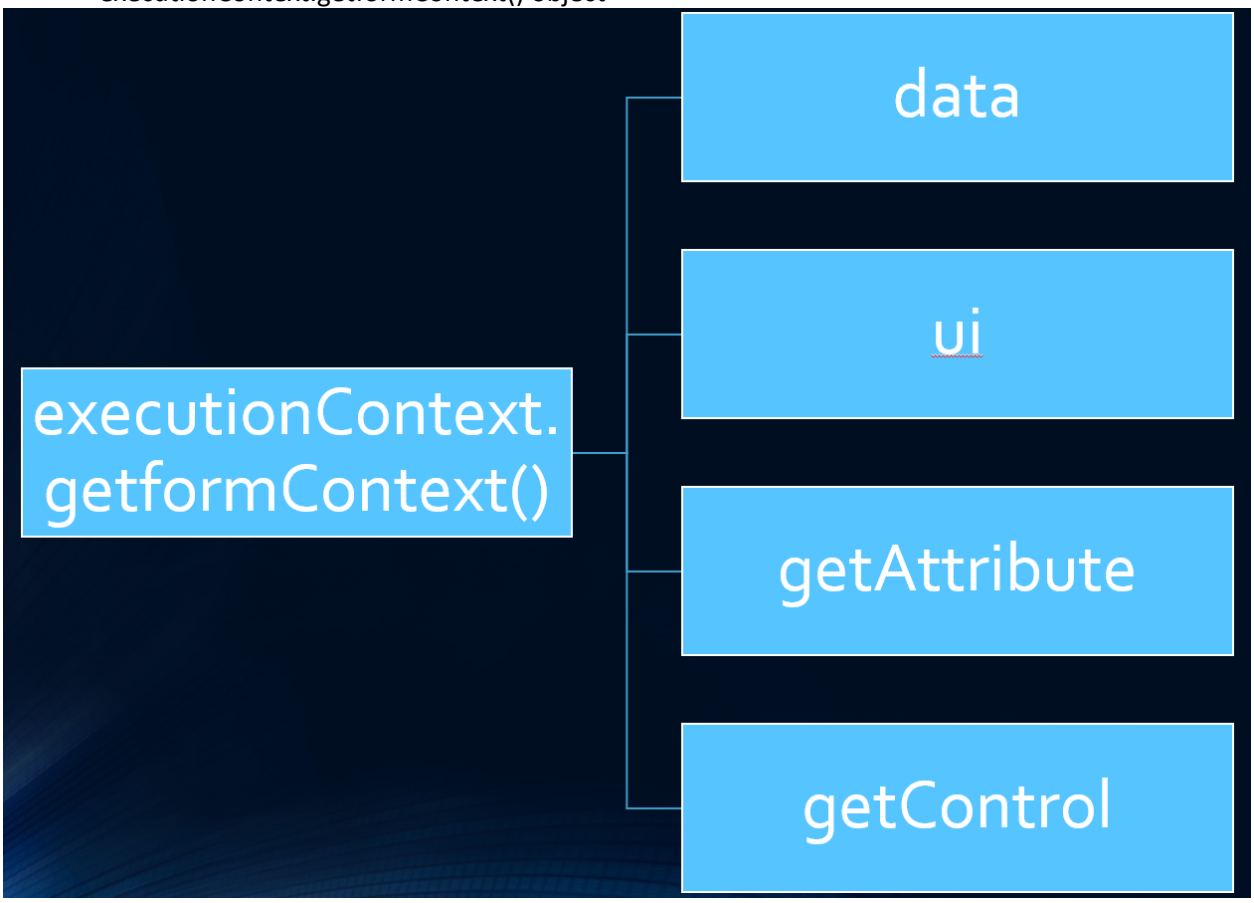
```
formContext.getControl("telephone1").addNotification({
        messages: ['Hello'],
        notificationLevel: 'RECOMMENDATION',
        uniqueID: 'IDUnique220912-2'
    });
```

    - clearNotification(uniqueId) removes the notification – for example:
        - formContext.getControl('name'). clearNotification('IDUnique220912-2')
    - getAttribute returns the column that the control is bound to.
    - getControlType returns a string which has the type of control (e.g. "lookup", "choices" or "standard")
    - get/setDisabled returns/sets whether a control is disabled (true if disabled)
    - get/setLabel returns/sets the control's label.
    - getName returns the control's name.
    - getOutput returns the control's output properties (as a dictionary)
    - getParent returns the section that the control is in.
    - get/setVisible returns whether the label is visible (true if visible).
    - setFocus sets the focus on the control
    - Additional methods may be available for lookup, choice and choices data types.
- Code created in this section:

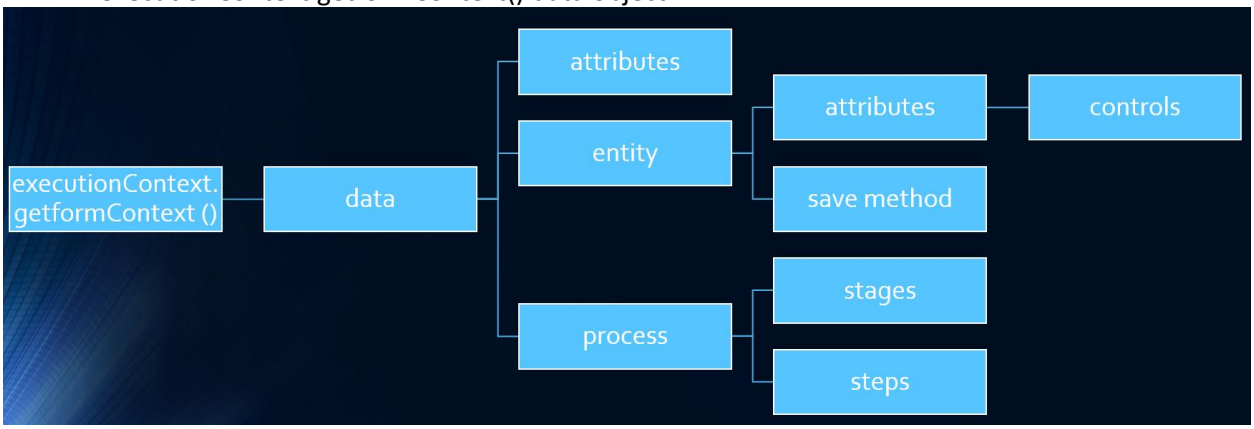
```
this.formOnLoad = function (executionContext)
{
    var formContext = executionContext.getFormContext();
    formContext.ui.setFormNotification("Hello world v6", "INFO", "IDUnique220912");
    if (formContext.getAttribute("fax").getValue() == null)
    {
        formContext.getAttribute("fax").setValue("123-4567");
        formContext.getControl("fax").addNotification({
            messages: ['Fax number set to default.'],
            notificationLevel: 'RECOMMENDATION',
            uniqueID: "IDUnique220912-2"
        });
        formContext.getControl("fax").clearNotification("IDUnique220912-2");
    }
}
this.AddressStreet3Hide = function (executionContext)
{
    var formContext = executionContext.getFormContext();
    if (formContext.getAttribute("address1_line2").getValue() == null)
    {

formContext.getControl("address1_composite_compositionLinkControl_address1_line3").set
Visible(false);
```

```
    } else {

formContext.getControl("address1_composite_compositionLinkControl_address1_line3").set
Visible(true);
    }
}
```

- executionContext.getformContext() object

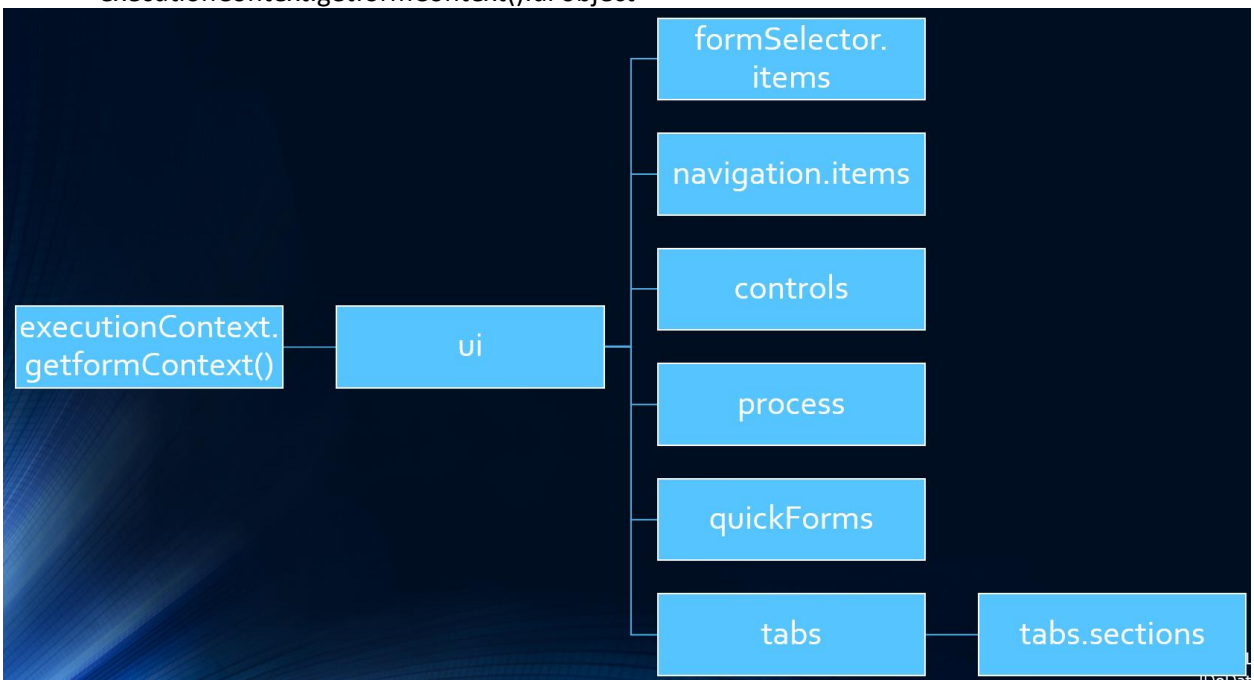


- executionContext.getformContext().data object



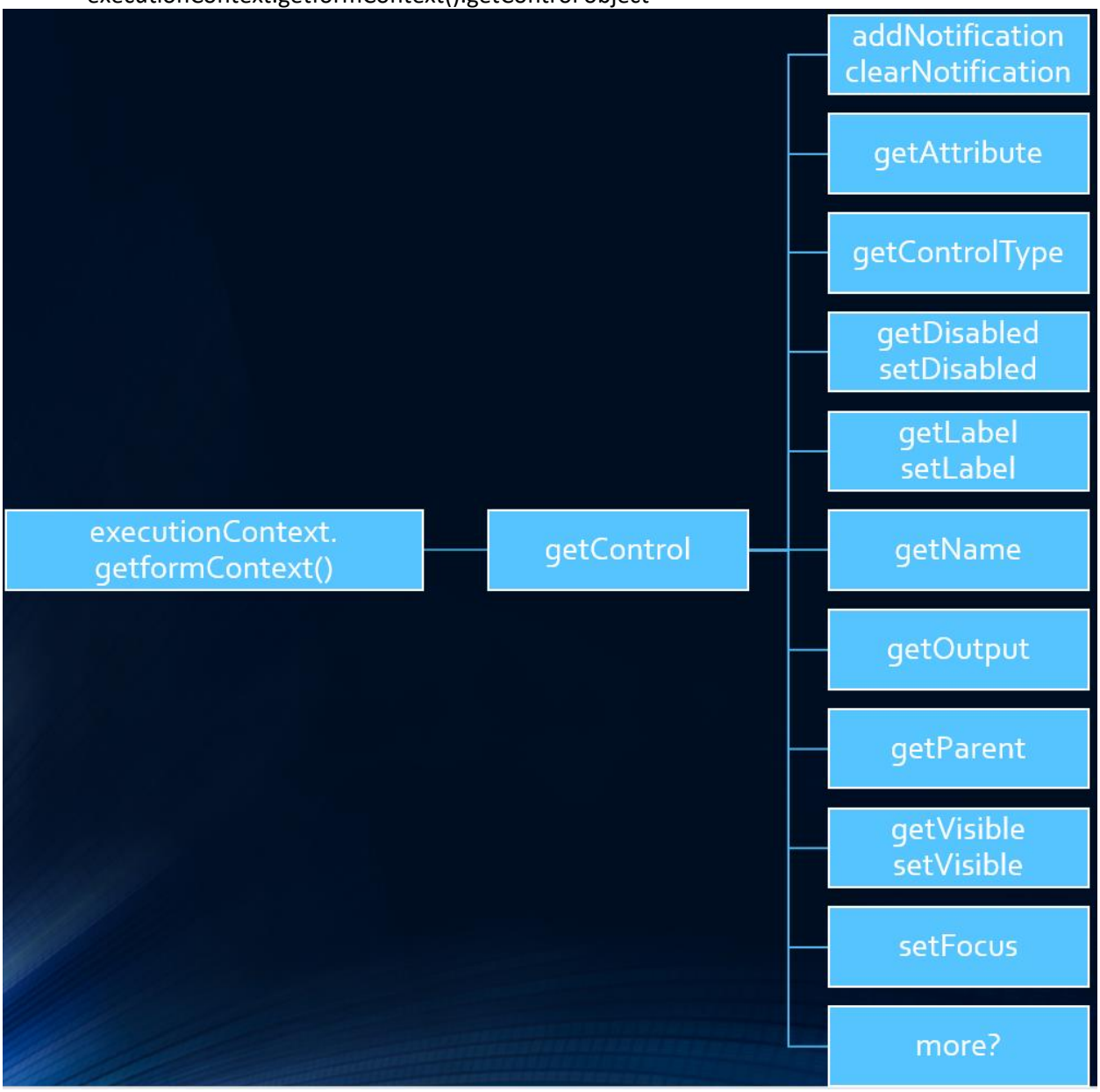
- addOnLoad and removeOnLoad
- getIsDirty
- isValid

    - refresh
    - save
- executionContext.getformContext().ui object



    - addOnLoad and removeOnLoad
    - close
    - getFormType – returns:
        - 0 = Undefined,
        - 1 = Create,
        - 2 = Update,
        - 3 = Read Only,
        - 4 = Disabled,
        - 6 = Bulk Edit.
    - getViewPortHeight and getViewPortWidth
    - setFormEntityName
    - setFormNotification(message, level, uniqueID)
    - clearFormNotification(uniqueID)

- executionContext.getformContext().getControl object



49. Register an event handler

- Event occur whenever:
  - A form loads
  - Data is changed in a column or a form item.
  - Data is saved in a form.
- JavaScript code can run in response to these events.
- To add/remove a form library (which contains the JavaScript):
  - For the Unified Interface:
    - In the Power Apps Portal, go to Data –Tables – [Select the table].
    - Go to Forms – [Click on/edit a form].
    - On the left-hand side, select Form libraries, then "+Add library", then "+New web resource".

        - Change "Type" to "JavaScript (JS)".
        - Upload the JavaScript file by clicking on "Choose file".
            - By doing it this way round, you will only see "JavaScript" in the "Types of files".
        - Enter a name and display name, then click OK.
        - Then go back into Form libraries, then "+Add library", and add the library you have just imported.
    - For Legacy:
        - In your model-driven app, go to the settings (gear) icon, then go to Advanced Settings, then Settings – Customizations, then "Customize the System".
        - In the Solutions explorer, go to Components – Web Resources, then click New.
        - Upload and add details as per the Unified Interface.
- To add/remove event handler functions to events in a form:
    - For the Unified Interface:
        - While editing the form, for form events:
            - On the right-hand side, go to the Events tab.
            - You can add "On Save" and "On Load".
        - For form fields [attributes]:
            - Click on the field.
            - On the right-hand side, go to the Events tab.
            - You can add "On Change".
    - For Legacy:
        - Go to the Form and go to Form Properties – Event Handler section.
    - You should click "Pass execution context as first parameter" to pass the execution context (and therefore access to formContext and gridContext) to the code.
    - There can be up to 50 event handlers for each event.

50. Create client-side scripts that target the Dataverse Web API

- Client-side events are:
    - Column
        - OnChange
    - Form
        - OnLoad
        - OnSave
    - Form data
        - OnLoad
    - Grid and subgrid
        - OnChange
        - OnLoad
        - OnRecordSelect
        - OnSave
    - IFRAME control
        - OnReadyStateComplete
    - Knowledge base search control
        - OnResultOpened
        - OnSelection
        - PostSearch
    - Lookup control

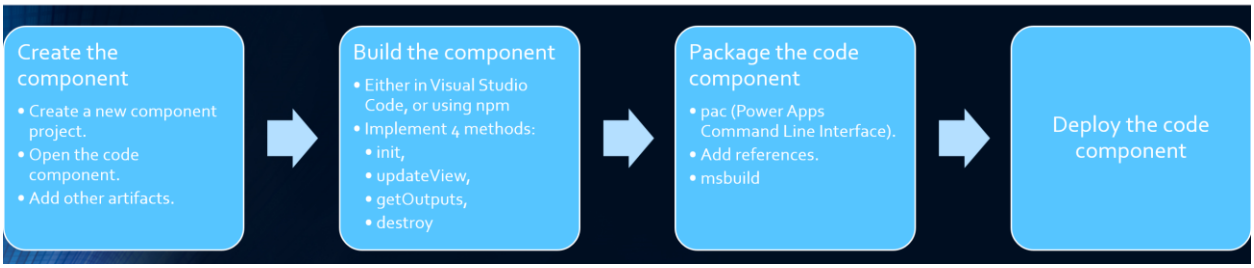
        - OnLookupTagClick
        - PreSearch
    - Process
        - OnProcessStatusChange
        - OnStageChange
        - OnStageSelected
    - Tab
        - TabStateChange

Create a Power Apps Component Framework (PCF) component

51. Describe the code component lifecycle

- What is a Power Apps component framework (PCF) control?
    - PCF allows you to create code components for:
        - model-driven apps (forms, views and dashboards),
        - canvas app screens, and
        - Power Pages (portals).
    - You can replace standard components with custom made, graphical components.
    - Unlike HTML web resources, it is loaded at the same time as other components.
    - All the HTML, CSS and TypeScript files can be bundled into a single solution package file and can be moved across environments.
        - You can also use React and Fluent libraries if you wish, to help create more attractive components.
    - You can have access to Web API, Utility and Device (audio/sound capture and GPS location) features.
- Process
    - Create the component
        - Create a new folder.
        - Navigate to the new folder.
        - Create a new component project.
            - It will also add all the required project dependencies.
        - Open the code component in a developer environment, such as Visual Studio Code.
        - Add other artifacts, such as manifest, component logic, styling.
            - Manifest is an XML file which describes the component's name, type of data that can be configured, properties and resources file names.
    - Build the component
        - Either in Visual Studio Code, or using npm
        - The index.ts file will implement 4 methods, which compose the code component lifecycle:
            - init (when it loads),
            - updateView (when app changes data),
            - getOutputs (when user changes data),
            - destroy (when the page where the component is on closes)
    - Package the code component
        - Use pac (Power Apps Command Line Interface) to create a new solutions project.
        - Add any references that may be needed.

        - Generate a zip file using msbuild
    - Deploy the code component
        - Directly from Power Platform CLI, or using Microsoft Power Platform Build Tools
- Describe the code component lifecycle



- Programs required
- Visual Studio Code (VSCode):
    - For Windows 8, 10, or 11, Linux, or macOS.
- Node.js
    - For Windows, Linux or macOS.
- Microsoft Power Platform CLI (Command Line Interface – PAC for short), either:
    - The standalone version
        - Needs Windows 10 or Windows 11
        - Launch it, and enter "pac install latest" to get the latest version.
    - Power Platform Tools for Visual Studio Code
        - Needs Windows 10, Windows 11, Linux, or macOS.
        - It updates automatically.
- .NET Build tools, using either:
    - Visual Studio 2022 for Windows & Mac,
    - Build Tools for Visual Studio 2022, or
    - Visual Studio 2019

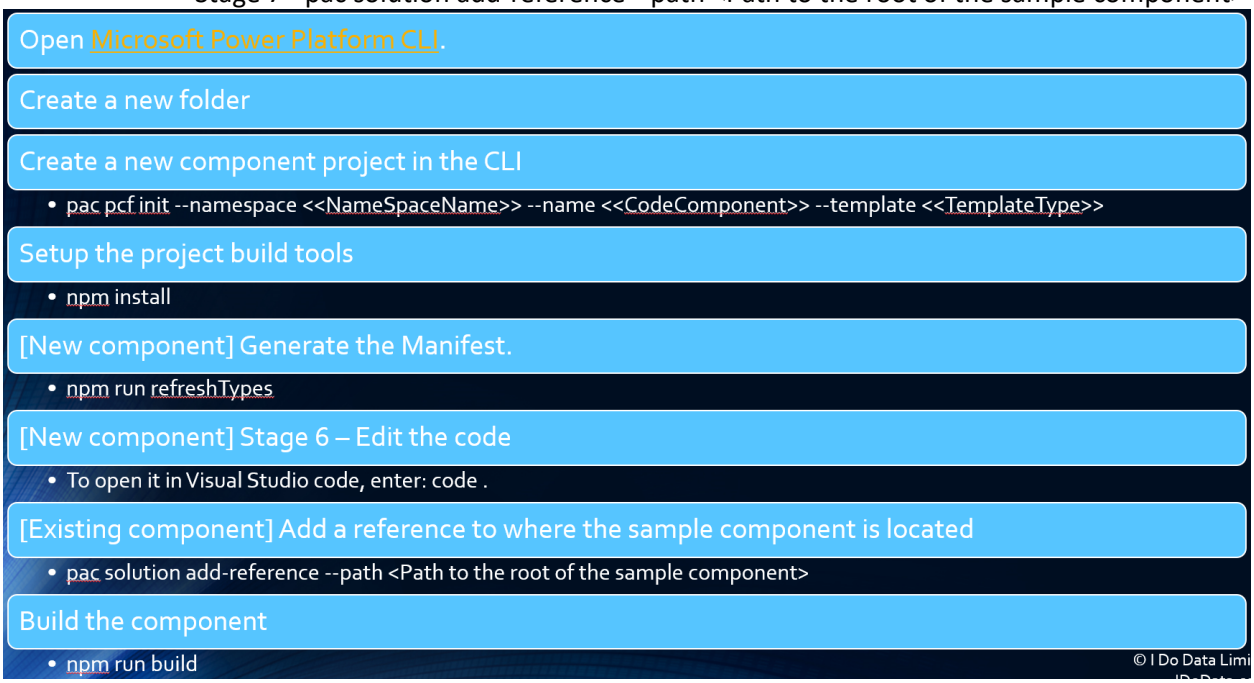
52. Initialize a new code component

- Stage 1 - Open Microsoft Power Platform CLI.
- Stage 2 - Create a new folder using
    - c:, d:, e: to change drives
    - cd (change directory)
        - cd "c:\My Directory" to go to an absolute folder.
        - cd "My Directory" to go to a subfolder.
        - cd .. to go back up a level.
    - md (change directory)
        - md "c:\My Directory"
- Stage 3 - Create a new component project in the CLI:

```
pac pcf init --namespace <<NameSpaceName>> --name <<CodeComponent>> --template
<<TemplateType>>
```

        - --namespace can be shortened to -ns.
        - --name can be shortened to -n.
        - --template can be shortened to -t.
        - TemplateType can be field or dataset.
- Stage 4 – The above may also run npm install to add the required project dependences.

        - If it doesn't, you need to run it.
        - npm is Node Package Manager. It puts modules where node can find them, and manages dependency conflicts.
        - If npm is "not recognised", then you need to install node.js.
- For a brand new component:
    - Stage 5 – You then need to generate the Manifest.
        - Run "npm run refreshTypes".
    - Stage 6 – Edit the code
        - To open it in Visual Studio code, enter code .
- For an existing component, add a reference to where the sample component is located:
    - Stage 7 - pac solution add-reference --path <Path to the root of the sample component>



53. Configure a code component manifest

- The component's metadata is contained in ControlManifest.Input.xml. It contains:
    - Control node – namespace, version and display information. It also contains:
        - property – data types,
        - data-set – a set of table records,
        - resources – details of resource files,
        - feature-usage – what features components should use.
- The manifest contains the **Control** node. This contains the following properties:
    - namespace – alphanumeric characters only,
    - constructor – the method for initializing the object, alphanumeric characters only
    - control-type [optional] – "standard" or "virtual" (a control using the React and platform libraries – currently in Preview),
    - description-key [optional] – the description of the code component (string),
    - display-name-key – the code component's name (string),
    - preview-image [optional] – image used on customization screens to show a preview of the component,
    - version – e.g. 1.0.0.

        - If you make any changes to the manifest, this must be updated in order to see any changes in the runtime.
- The manifest also contains the **property** node, which contains:
    - name – property's name,
    - description-key [optional] – localized property's description
    - display-name-key – localized property's name,
    - of-type or of-type-group – the property's data type. Use of-type-group if you want to have two or more than data types. Data types include:
        - Strings
            - Multiple (lines of text, up to 1,048,576 characters),
            - SingleLine.Email, .Phone, .Text, .Ticker and .URL
            - SingleLine.TextArea (multiple lines of text, up to 4,000 characters),
        - Numbers
            - Currency – values between -/+922,337,203,685,477,
            - Decimal between -/+100 billion,
            - FP (floating point),
            - Whole.None (a whole number).
        - Dates
            - DateAndTime.DateAndTime,
            - DateAndTime.DateOnly,
        - Others
            - Lookup.Simple (a custom lookup),
            - MultiSelectOptionSet (choices),
            - OptionSet (choice),
            - TwoOptions (Yes/No)

```
<type-group name="numeric">
<type>FP</type>
<type>Decimal</type>
<type>Whole.None</type> </type-group>
```

    - usage - either
        - "bound" (the property represents a columns that the component can change), or
        - "input" (read-only values)
    - default-value [optional] – only used for "input" properties.
    - required [optional] – Boolean, whether the property is required or not.
- The manifest also contains the **resources** element. It includes:
    - code – the "path" where resource files are located. Also include the "order" in which the resource files should load.
    - css – Cascading Styles Sheets. Includes path and order.
    - img – Images. Includes path.
    - resx – localized strings. Includes path and "version"
    - platform-library – used in React controls and platform libraries.
- The manifest also contains the **dataset** node. It includes:
    - name, description-key and display-name-key,
    - cds-data-set-options – if "true", displays the Commandbar, ViewSelector and QuickFind.
- The manifest also contains the **feature-usage** node. It includes the following "uses-features":
    - Device.captureAudio, .captureImage, .captureVideo, .getBarcodeValue, .getCurrentPosition and .pickFile.
    - Utility, and WebAPI.

54. Implement component interfaces

- There are 4 methods for a standard component in the index.ts file:
  - init
    - Initializes the component instance
    - Can start remote server calls.
    - Cannot initialize dataset values – that can be done in the updateView method.
    - Parameters:
      - context – Input Properties containing parameters, component metadata and interface functions.
      - notifyOutputChanged – a method which notifies the framework that it has new outputs
      - state – component state that is saved from setControlState in the last session.
      - container – the HTMLDivElement that is to be rendered.
  - updateView
    - Called when any value in the property bag has changed:
      - field values,
      - datasets,
      - global values like container height and width,
      - offline status,
      - component metadata values such as label and visible.
    - Parameter – "context" contains values as defined in the manifest, and the utility functions.
  - getOutputs
    - Called prior to a component receiving new data.
    - Returns an object as defined in manifest.
  - destroy
    - Called when the component is to be removed from the DOM tree.

```
import {IInputs, IOutputs} from "./generated/ManifestTypes";

export class PL400component implements ComponentFramework.StandardControl<IInputs,
IOutputs> {

private myNotifyOutputChanged: () => void;

private myMainDiv: HTMLDivElement;

private myTextbox: HTMLTextAreaElement;

private myLabel: HTMLLabelElement;

private myButton: HTMLButtonElement;

private myButtonHandler: EventListener;

private myisUpperCaseOnly: boolean;

private myTextboxHandler: EventListener;

    constructor()

    {
```

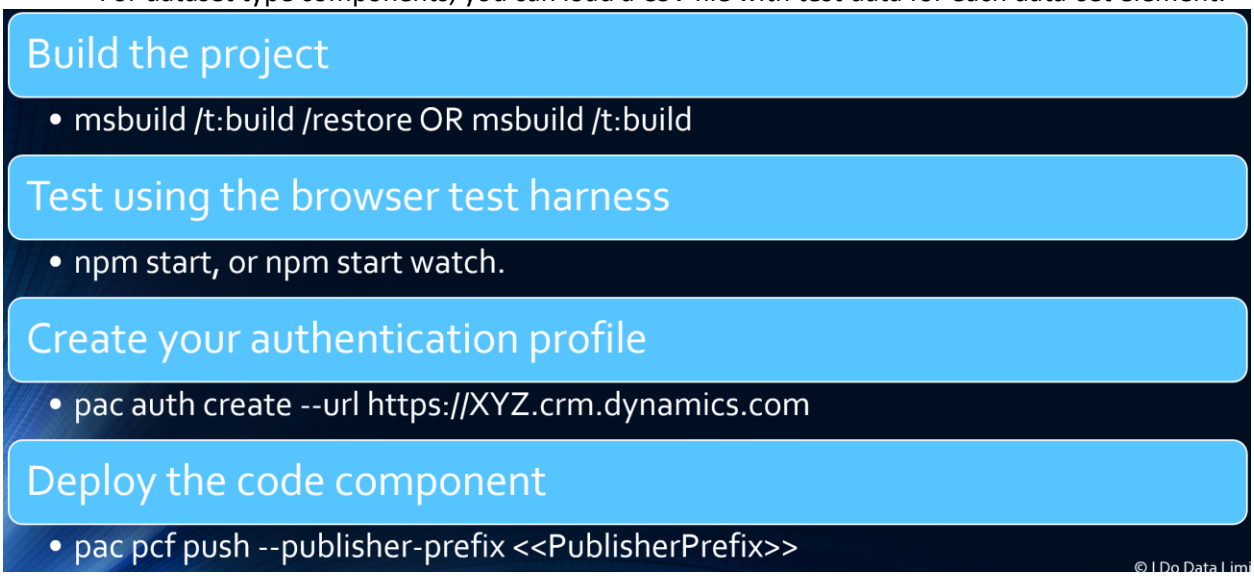
```
    }

     */
    public init(context: ComponentFramework.Context<IInputs>, notifyOutputChanged: ()
=> void, state: ComponentFramework.Dictionary, container:HTMLDivElement): void
    {
        this.myNotifyOutputChanged = notifyOutputChanged;
        this.myMainDiv = document.createElement("div");

        // This creates myTextbox and a text box
        this.myTextbox = document.createElement("textarea");
        this.myTextbox.value = context.parameters.textValue.raw || "";
        this.myMainDiv.appendChild(this.myTextbox);
        this.myTextboxHandler = this.myTextBoxHasChanged.bind(this);
        this.myTextbox.addEventListener("input", this.myTextboxHandler);

        // This creates label
        this.myLabel = document.createElement("label");
        this.myMainDiv.appendChild(this.myLabel);
        this.myisUpperCaseOnly = context.parameters.isUpperCaseOnly.raw || false;

        // This creates button
        this.myButton = document.createElement("button");
        this.myButton.textContent = "Click me";
        this.myButtonHandler = this.myButtonClicked.bind(this);
        this.myButton.addEventListener("click", this.myButtonHandler);
        this.myMainDiv.appendChild(this.myButton);

        // This adds everything into container
        container.appendChild(this.myMainDiv);
    }

    public myTextBoxHasChanged()
    {
        this.myNotifyOutputChanged();
    }

    public myButtonClicked()
    {
        this.myisUpperCaseOnly = !this.myisUpperCaseOnly;
        if (this.myisUpperCaseOnly)
        {
            this.myLabel.innerHTML = "UPPER CASE ONLY";
            this.myTextbox.value = this.myTextbox.value.toUpperCase();
```

```
        } else if (!this.myisUpperCaseOnly && this.myLabel.innerHTML != "UPPER/lower
case")
        {
            this.myLabel.innerHTML = "UPPER/lower case";
        }

        // Calls NotifyOutputChanged
        this.myNotifyOutputChanged();
    }

    public updateView(context: ComponentFramework.Context<IInputs>): void
    {
        this.myTextbox.value = context.parameters.textValue.raw || "";
        this.myisUpperCaseOnly = context.parameters.isUpperCaseOnly.raw || false;


        if (this.myisUpperCaseOnly)
        {
            this.myLabel.innerHTML = "UPPER CASE ONLY";
            this.myTextbox.value = this.myTextbox.value.toUpperCase();
        } else if (!this.myisUpperCaseOnly && this.myLabel.innerHTML != "UPPER/lower
case")
        {
            this.myLabel.innerHTML = "UPPER/lower case";
        }

        // Calls NotifyOutputChanged
        this.myNotifyOutputChanged();
    }

    public getOutputs(): IOutputs
    {
        return {
            textValue: this.myTextbox.value,
            isUpperCaseOnly: this.myisUpperCaseOnly
        };
    }

     */
    public destroy(): void {}
}
```

55. Package, deploy, and consume a component

- To package a component:
  - Build the project.
    - The first time, use: msbuild /t:build /restore

    - Other times, use: msbuild /t:build
        - "/t" is short for "target". "/restore" runs the Restore target prior to building the actual targets. "build" is the main build entry point.
    - Once it is built, the solution files are located inside the \bin\debug\ folder.
- To this it, use the browser test harness
    - npm start , or
    - npm start watch.
    - The latter allows you to make changes to the index.ts file, any imported modules, and all resources listed in the ControlManifest.Input.xml (but not the ControlManifest file itself) and see the results in real-time (but you need to save it first!).
        - If you make any changes to the ControlManifest file itself, you will need to rebuild the project.
- The browser test harness includes:
    - Context Inputs includes:
        - Form Factor – web, tablet and phone, so you can see the component on the different form factors.
        - Component Container Width and Height
    - Data Inputs – all the properties and their types/type-groups in the manifest file.
    - Outputs – the output whenever a component's getOutputs method are called.
- For dataset type components, you can load a CSV file with test data for each data-set element.



- You can then manually import it, or you can use Microsoft Power Platform CLI.
- To connect to your component:
    - Create your authentication profile:
      ```
      pac auth create --url https://XYZ.crm.dynamics.com
      ```
        - You can find XYZ by going to the Power Apps portal and go to Advanced settings. It is in the URL.
        - You will then connect to your environment.
    - You can view existing profiles by using:
      ```
      pac auth list
      ```
    - You can switch between other authentication profiles, or delete a profile, by using:
      ```
      pac auth select/delete --index <<index>>
      ```
    - You can find your environment by using:

```
pac org who
```

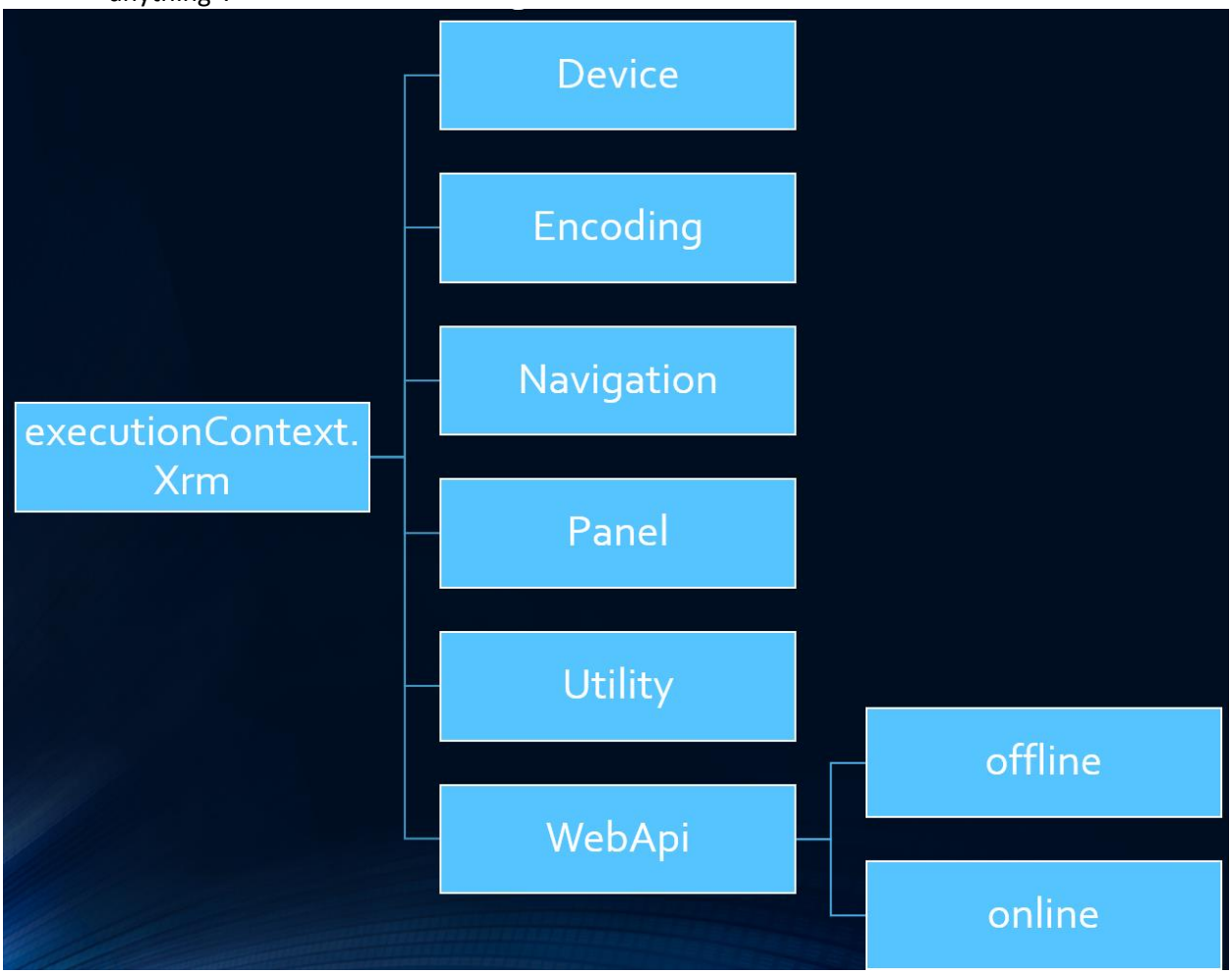
- To deploy the code component:
  - Make sure you have a valid authentication profile.
  - Navigate to the relevant directory.
  - Run the command:

    ```
    pac pcf push --publisher-prefix <<PublisherPrefix>>
    ```

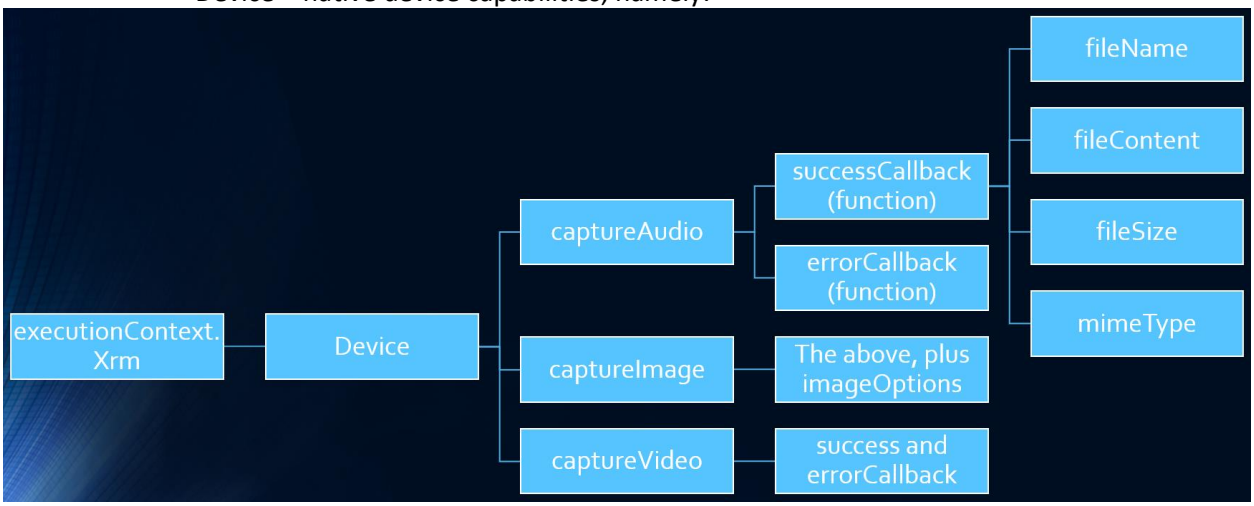
    - The Publisher Prefix should match that in which the components will be included.
- To load a component to a model-driven app column:
  - Go to Dataverse – Tables – [the relevant table] – Forms – edit the Main form.
  - Go to Components (on the left-hand side) – Get more components (at the bottom), and then add the component.
  - Then go to Components – More components, and click on your component onto the screen.
  - Bind any columns as appropriate, and select whether this control should be viewed on the Web, Phone or Tablet.
- To load a component to a canvas app:
  - First of all:
    - Go to Power Apps – Settings – Admin Center – Environments – [relevant Environment] – Settings – Products – Features.
    - Turn on "Power Apps component framework for canvas apps".
    - Click Save.
    - This only needs to be done once per environment.
  - Go to the canvas app in edit mode.
  - Click on "+ Add", and "Get more components".
  - In the "Import components" screen, click on Code, select the component, and click Import.
  - Click on "+ Add", go to Code components and select the component.
  - On the Properties tab, you will see any exposed properties.
- If you want to remove the component:
  - go to click on the Tree view – Screens, click on the … next to the Screen and click on Delete.
- If you want to update the code component, make sure you update the version property in the manifest file.
  - Make sure you close the app and reopen it to see the updated code component.

56. Configure and use Device, Utility, and WebAPI features

- Xrm is an acronym derived from CRM (Customer Relationship Management). X means "anything".

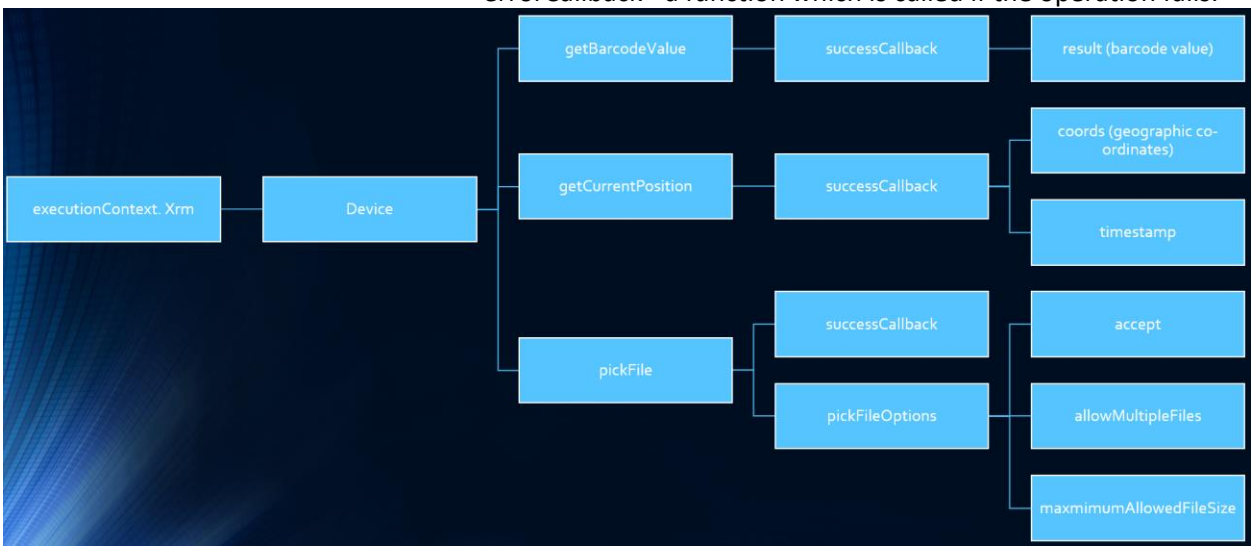


- executionContext.Xrm contains:
  - Device – native device capabilities, namely:

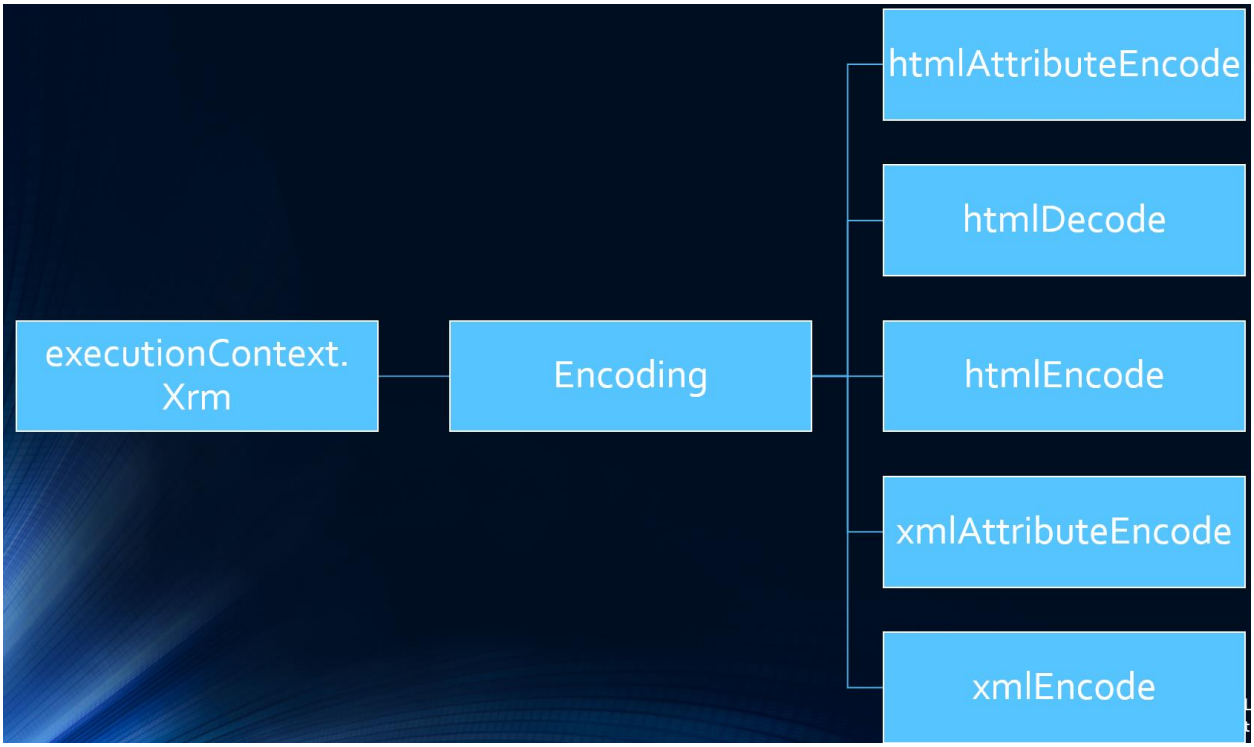


  - captureAudio, captureImage, captureVideo and pickFile:

- captureAudio: microphone audio recording.
- captureImage: camera image capture. This includes options parameter, allowing for:
    - allowEdit (Boolean),
    - preferFrontCamera,
    - quality (as a percentage), and
    - height and width
- captureVideo: camera video recording,
- pickFile: dialog box to choose files to upload. This includes the options parameter, allowing for:
    - accept – which file types do you want (audio, video or image)?
    - allowMultipleFiles, and
    - maximumAllowedFileSize
- They all use:
    - successCallback exposes filename, .fileContent, .fileSize and .mimeType
    - errorCallback - a function which is called if the operation falls.

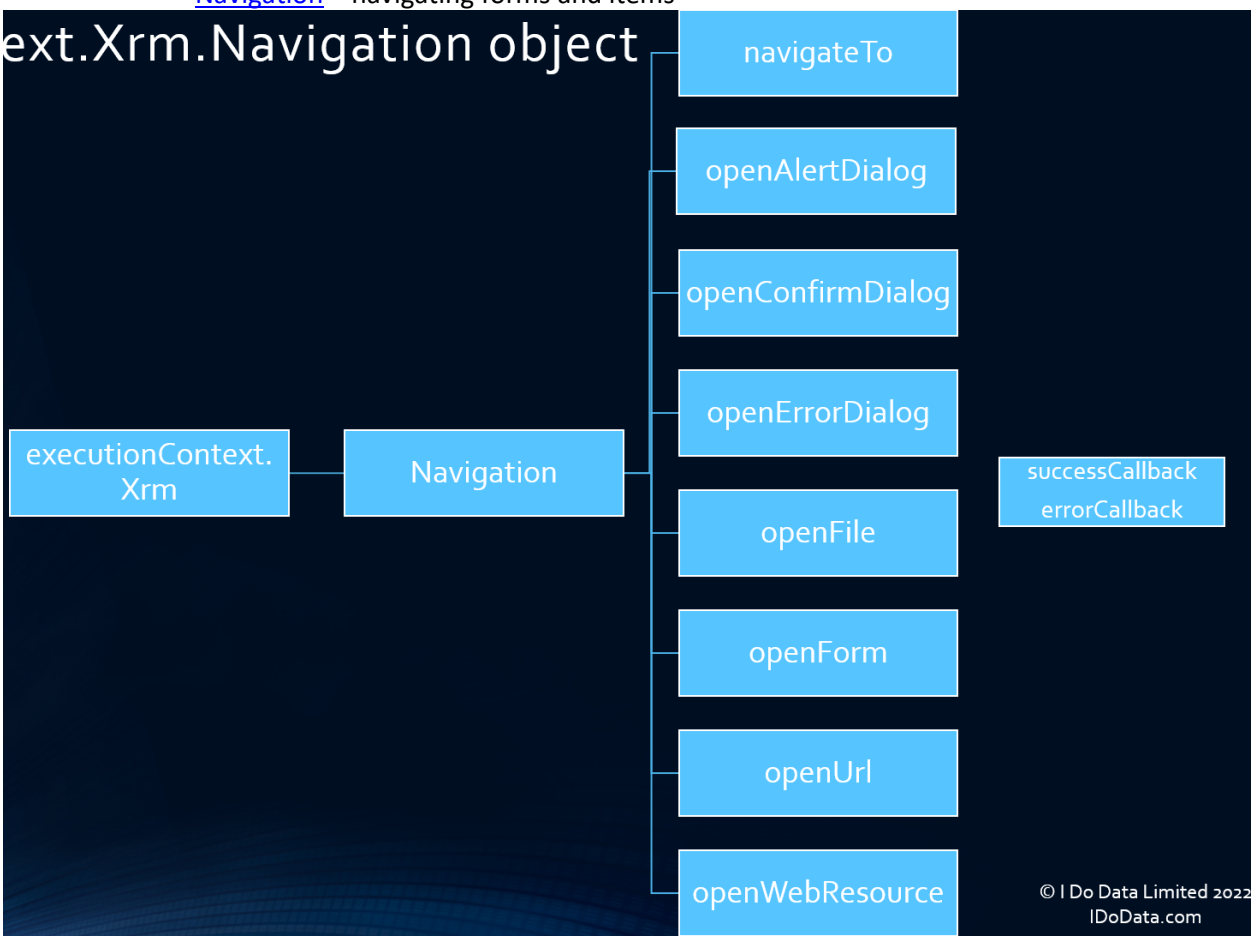


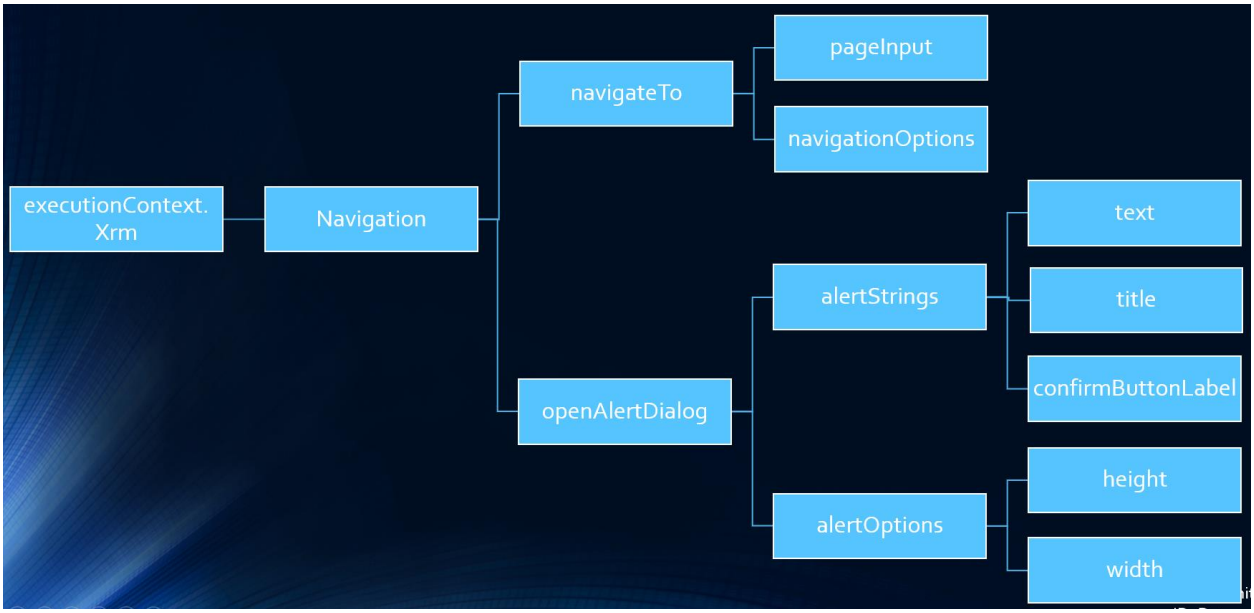
- getBarcodeValue: camera barcode scanning. Uses:
    - errorCallback, and successCallback – barcode value returned as a String
- getCurrentPosition: device's geolocation. Uses:
    - errorCallback and successCallback – coords and timestamp



- Encoding – encoding strings to and from HTML and XML
    - htmlAttributeEncode, htmlDecode, htmlEncode,
    - xmlAttributeEncode, xmlEncode

- Navigation – navigating forms and items



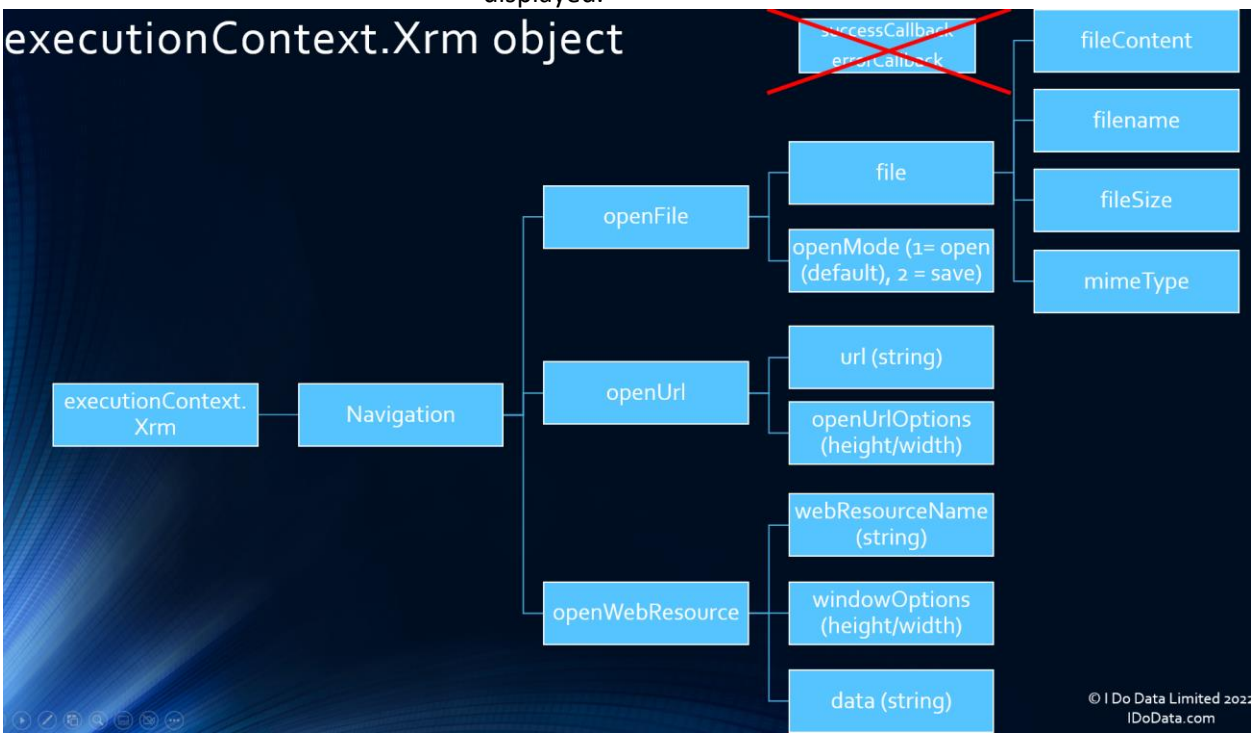


- navigateTo navigates to a table list or record, HTML web resource, or custom page. Parameters:

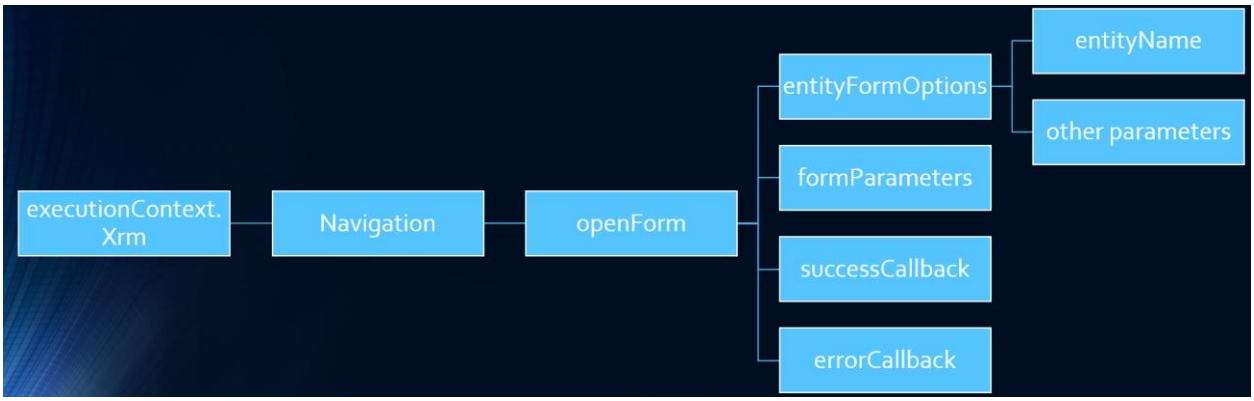
        - pageInput – information about the page, which could be an entity list or record, dashboard, HTML web resource, or custom page.
        - navigationOptions – whether it is opened inline or in a dialog
    - openAlertDialog opens a dialog box with a message and 1 button.
        - Example:
            - context.navigation.openAlertDialog({text:"Hello", confirmButtonLabel : "Yes"}, {height: 100, width: 200})
        - Parameters:
            - alertStrings – this has:
                - "text", the message, and
                - "confirmButtonLabel", the button label ("OK", in the user's preferred language, is the default).
            - options – this has "height" and "width" in pixels.

    - openConfirmDialog to open dialog boxes with a message and 2 buttons.
        - Parameters:
            - confirmStrings – this has:
                - confirmButtonLabel and cancelButtonLabel (localized versions of "OK" and "Cancel" are the default),
                - "title", "subtitle" and "text" – to be used in the dialog box.
                - options – this has "height" and "width" in pixels.
        - The successCallback function includes the "confirmed" attribute. The Boolean attribute shown whether the confirm button had been clicked.

- openErrorDialog opens an error dialog.
  - Parameters – options has:
    - "details" (optional) – this enables the "Download Log File" button, and clicking this downloads a text file with the "details".
    - "errorCode" – a Number, and "message" – string. One of these most be set. If errorCode, a default error message will be displayed.



- openFile, openForm, openUrl and openWebResource opens these items.
  - openFile(file, options).
    - "file" contains fileContent (string), filename (string), fileSize (Number – in Kb) and mimeType (string).
    - "options" – 1 = open, 2 = save. (1/open is default).
  - openUrl contains: "url", and "options", which contains "height" and "width"
  - openWebResource contains: "name" (the HTML web resource), "options", which contains "height", "width" and "openInNewWindow", and "data" to be passed to the Web Resource.



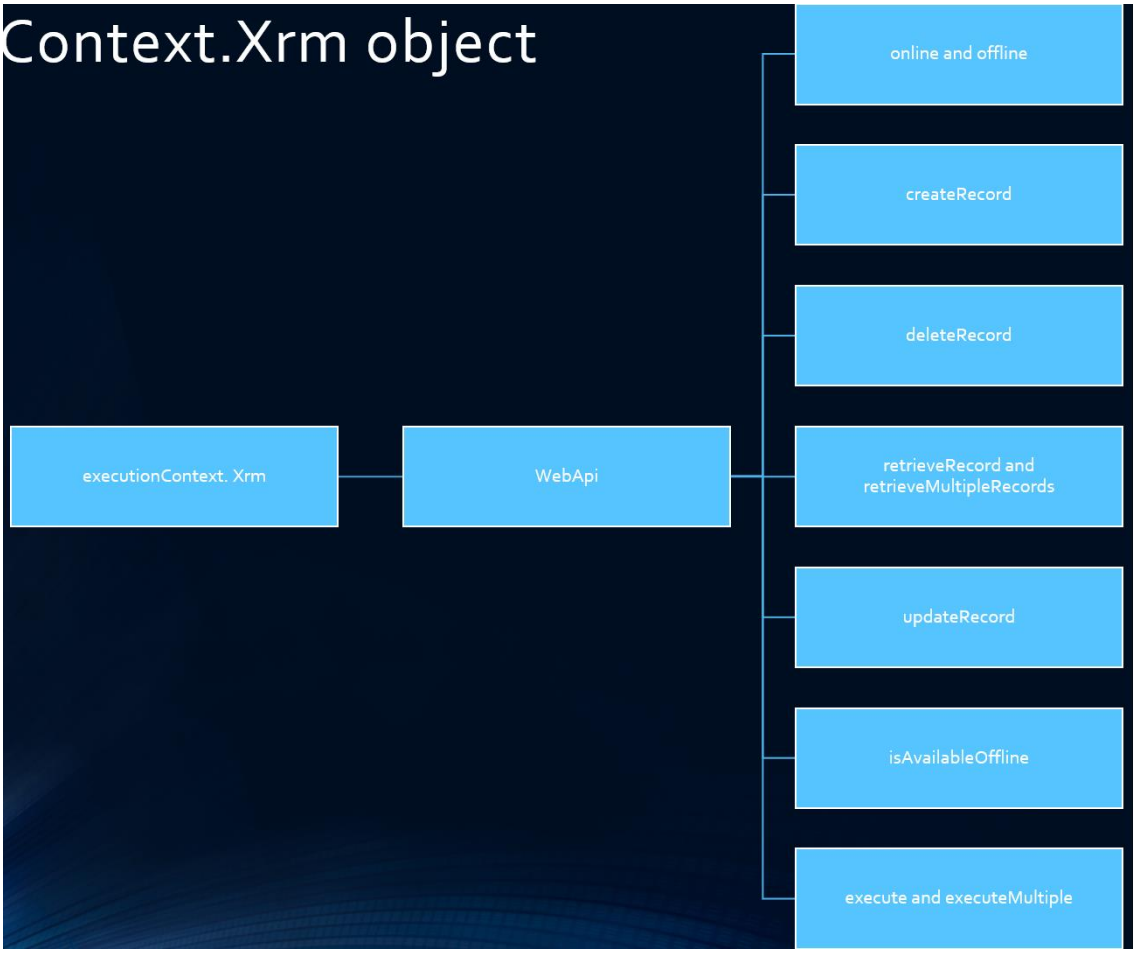
- openForm(options, parameters)
  - "options" contains:
    - createFromEntity – a Lookup which designates a record that provides default values based on mapped attribute values. Contains "entityType", "id" and "name".
    - entityID – ID of the table record for which to display the form.
    - entityName – Logical table name for the form.
    - formID – Form instance ID (string)
    - height and width – number.
    - openInNewWindow – if True, need to specify height or width. Otherwise, displays in a new tab.
    - useQuickCreateForm – whether to open a Quick Create Form. By default, false.
    - windowPosition – 1 = center, 2 = side of screen.
  - "parameters" contain extra parameters for the form.
- Panel – displays a web page in the side pane of a model-driven app form.
  - loadPanel - Opens a web page in a side pane of a model-driven app web client. Preview feature – don't use in production.

- Utility contains:

- show and closeProgressIndicator – open/removes progress dialog
    - showProgressIndicator has the Parameter "message", which is displayed in the progress dialog.
    - The progress dialog blocks the UI until it is closed.
- getAllowedStatusTransitions – valid status transitions
    - entityName (string),
    - statusCode (status code number),
    - then(successCallback, errorCallback) – code to run when the operations succeeds/fails.
- getEntityMetadata – returns definition for a table
    - entityName, attributes – array of strings with columns in table
    - then(successCallback, errorCallback)
- getEntityMainFormDescriptor – gets default main form description for a table
    - entityName, formId (the form ID)
- getGlobalContext – gets global Context. Properties are:
    - client – such as
        - getClient (Web, Outlook, Mobile),

        - getFormFactor (0-3 = Unknown, desktop, tablet, phone),
        - isOffLine (whether the server is offline or not),
        - isNetworkAvailable
    - organizationSettings – such as
        - baseCurrency, defaultCountryCode, languageID and uniqueName (of the current organization)
    - userSettings – such as
        - dateFormattingInfo, languageId, roles and securityRoles, transactionCurrency, userId, username and getTimeZoneOffsetMinutes
- getLearningPathAttributeName – for Learning Path
- getPageContext – the page object. Returns:
    - pageType (either entityrecord or entitylist) and entityName,
    - If pageType = "entityrecord", then entityId, createFromEntity and formId
    - If pageType = "entityName", then viewId and viewType.
- getResourceString – localized string
    - webResourceName – web resource name, and
    - key – localized string key
- invokeProcessAction – custom action
- lookupObjects – opens a lookup control
- refreshParentGrid – refreshes the parent grid with a particular record

- WebApi – create and manage records, and run Web API actions and functions, either online or offline.



- Properties: online and offline.
  - var onlineWebApi = Xrm.WebApi.online;
- Methods:
  - createRecord, delete, retrieve and updateRecord.
    - Xrm.WebApi.createRecord(entityLogicalName, data).then(successCallback, errorCallback);
      - entityLogicalName – logical table name.
      - data – JSON defining columns and values.
      - then(successCallback, errorCallback) – function to call when operation succeeds/fails
    - Xrm.WebApi.deleteRecord(entityLogicalName, id).then(successCallback, errorCallback);
      - id – table record GUID to delete.
    - Xrm.WebApi.retrieveRecord(entityLogicalName, id, options).then(successCallback, errorCallback);
      - options – OData system query options
        - $select - to limit the properties return, and
        - $expand – to control what data from related tables is returned.

            - For example: $select=name&$expand= primarycontactid($select=contactid,fullname)
        - retrieveMultipleRecords
        - Xrm.WebApi.createRecord(entityLogicalName, id, data).then(successCallback, errorCallback);
        - isAvailableOffline
        - execute and executeMultiple
            - Executes one or a collection of action, function or CRUD operations. Only available online

Create a command button function

57. Create a command function - Modern

- In the modern interface, edit the model-driven app:
    - By going to Power Apps – Solutions - [go to the relevant solution] – click on the … next to the app – Edit.
- In the modern designer:
    - Go to Pages (on the left-hand side),
    - Click on the … next to the relevant table (not form or view), and select "Edit command bar".
- Select the relevant command bar:
    - Main grid – shown when using the left-hand navigation of an app to view a full page list of records in a table.
    - Main form – shown when viewing an individual row.
    - Subgrid view – for example, if you view a Contact form, you might see within it an Email subgrid showing all emails for that particular contact.
    - Associated view – shown when a grid of related records appears in the navigation pane.
    - You can change which command bar you are editing using the drop-down at the top.
- You will then see the commands.
    - On the left-hand side in a tree view, and
    - In the canvas
- You can click on +New to add:
    - A dropdown – a menu
    - A split button – allows for a menu to be shown or a command executed
    - A group – add bold titles for commands nested within dropdowns and split buttons.
    - A command.
        - This standard button was called flyouts in the classic interface.
        - Performs an action when selected.
        - Can be nested in groups with dropdowns and split buttons.
- In the classic interface, creating commands requires editing XML files within solution files or using 3rd party tools.
- One such 3rd party tool is Ribbon Workbench within XRM Toolbox.
- It is recommended that you use a blank solution.
    - It will not work with solutions which have more than 5 tables.
    - Then add it in the relevant table.
    - As the changes happen at the table level, it will be reflected in that table as used in other solutions.
- Connect to your environment

Create a command button function

- Open Ribbon Workbench in XRM Toolbox.
- Connect to the organization, then select the solution.
- Drag a Button from the Toolbox to the Home, SubGrid, or Form.
- Add an ID.
- Go to Commands (Actions).
    - Add the ID (the IDCommand)
    - Add the JavaScript action:
        - Library (.js file),
        - Function Name (full name, including function name).
        - You can also pass in Parameters.
    - Add Display Rules and Enable Rules (when the ribbon control is shown or enabled).
- Go back to the control.
    - In Command, add the IDCommand.
    - Add Label, Alt, Tool Tip Title and Description.
    - Add ModernImage (a .svg) file.
- Click Publish.
    - Take a backup first.
- Go back into the App, and the button will be there.

58. Design command button rules and actions - Modern

- Using the left-hand pane, you can:
    - Move up/down/inside/out,
        - You can also go to the canvas (not the left-hand pane, but the canvas) and drag it to a new location.
    - Duplicate,
    - Copy/Paste,
    - Hide and
    - Delete
- Commands
    - The Label is what is displayed on the command button.
    - Icon:
        - You can use a system item, or
        - upload or use a web resource SVG file
    - The Action is what runs when you click on this button.
        - You can use Power Fx or
        - JavaScript. You need to:
            - Select or add a library,
            - Enter a function name, and
            - Add any parameters, if needed
    - You could change the Visible (which used to be known as "rules") to define whether the button should be shown or not.
        - However, by default, if you have a grid with an item selected, all custom buttons are hidden.
    - The optional Tooltip title and descriptions are displayed to the user if they hover over the command.
    - The optional Accessibility text will be read by screen readers.

    - The order number is the order the commands will be displayed, in relation to other commands in the command bar.
- You can write coding using Power Fx:
    - An example is changing the Action to:
        - Notify("Hello world",NotificationType.Success)
            - Make sure you use capital letters as appropriate. Use the auto-complete and press Tab to ensure you have the correct capitalization.
    - Self. references the current control. It also has the Selected property, which has the following properties:
        - Self.Selected.Item refers to one of the records selected from the DataSource.
            - If no record is selected, then this returns blank. You can test this using IsBlank.
        - Self.Selected.AllItems refers to all of the records selected from the DataSource.
            - If no record is selected, then this returns an empty table. You can test this using IsEmpty.
        - Self.Selected.State shows the state of the selected control: 0=Edit, 1=New and 2=View.
        - Unsaved returns true if Selected or SelectedItems have unsaved changes.
            - If AutoSave is true, then it will always return false.
    - So, you could set Visibility in a Main form to:
        - Self.Selected.Item.'Address 1: City'="Redmond"
            - You cannot do this in a grid, as custom buttons are hidden when something is selected!
    - To update the current record, use:
        - Patch(Accounts,Self.Selected.Item, {address1_city:"New Redmond"})
            - To get the "address1_city", in the Power Apps portal, go to Dataverse – Tables – [click on the relevant table] – Columns – [click on the … next to the relevant column] – Tools – Copy logical name, or look at the Name column.
    - To add a new record, use:
        - Patch(Accounts,Defaults(Accounts), {'Account Name': "New Name"}, {'Address 1: City':"New Redmond"})
    - You can navigate to a custom page or a default view of a table, using
        - Navigate(Accounts)
    - If you want a confirmation dialog box, use Confirm
        - If(Confirm("message", {Title: "Title"},Notify("Yes"))
        - Confirm allows for:
            - Title – but don't make this too long.
            - These currently don't seem to work in model-driven apps.
                - ConfirmButton – by default, this is OK.
                - CancelButton – by default, this is Cancel.
                - Title and Subtitle – don't make this too long.
    - To load a webpage, use Launch
        - Launch("http://www.Microsoft.com");
- A basic function is:

```
function ButtonPress(primaryControl)
{
    var formContext = primaryControl;
```

```
    Xrm.Navigation.openAlertDialog({
        text: "Click!"
    });
}
```

- A more complex function is:

```
function ButtonPress(primaryControl)
{
    var formContext = primaryControl;
    Xrm.Navigation.openAlertDialog({
        text: "Click!",
        confirmButtonLabel: "Button label",
        title: "Dialog box for ".concat(formContext.getAttribute("name").getValue())
    }).then
    (function (success)
    {
        Xrm.Navigation.openConfirmDialog({
            text: "You have successfully clicked."
        })
    }
    );
    formContext.getControl("name").setLabel("name");
}
```

59. Manage dependencies between JavaScript libraries

- To view dependencies:
  - Go to an item in a Solution.
  - Go to Advanced – Show dependences.
- It shows:
  - What items are dependent on this item, and
  - What this item is dependent on.
- For example:
  - a RESX (string) web resource is dependent on a JavaScript web resource, or
  - a JavaScript web resource is dependent on a second JavaScript web resource.
- Dependencies are established after the resource has been configured and published.
- To add a dependency,
  - Click on the … next to the Web Resource in the solution.
  - Go to Edit – Edit in classic.
  - Go to the Dependencies tab.
  - Add any dependencies.
- The purposes for doing this are:
  - You cannot delete a solution component which is dependent on another one.
  - In model-driven apps, if you have a JavaScript web resource which depends on another web resource, both will be automatically loaded. Otherwise, you may find that the web resource fails because its dependent web resource is not loaded.
    - You do not need to explicitly load them.
    - They are loaded asynchronously and in parallel.

- What are plug-ins?
- Plug-ins can run .NET code uploaded to the Microsoft Database.
- Advantages of plug-ins:
    - they can perform well, and
    - they can be powerful.
- Disadvantages of plug-ins:
    - they require developer skills, and
    - they can be written poorly and impact on performance.
- Consider alternatives:
    - Power Automate and workflows allow you to create automated actions, potentially using code in multiple flows or workflows.
    - Calculated and rollup fields are already incorporated in Power Platform.
    - Custom Actions can create re-usable messages called from other workflows or from web service endpoints.
    - Azure Service Bus integration and Webhooks can push data to external systems, where logic can be applied using different resources (see topic 82).

Create a plug-in

60. Describe the plug-in execution pipeline

- Plug-ins can run .NET code uploaded to the Microsoft Database, and is based on events (data operations) in the Dataverse platform.
    - These events are preceded and followed by messages.
    - The plug-ins "listen" for these messages, and run at the appropriate time.
    - The events are:
        - Create – creating a record.
        - Retrieve – retrieving a record.
        - RetrieveMultiple (one word) – retrieving a collection of records
        - Update – updating an existing record
        - Delete – deleting a record
        - Associate – creating a link between records
        - Disassociate – deleting a link between records
    - We will be using something called "steps" in the Plug-in Registration tool to associate code with a message.
- The messages happen during the Event Pipeline Stage of Execution.
    - PreValidation
        - This happens before the main system operation, before any security checks, and before the data transaction.
        - You can then cancel the data transaction if necessary at this stage by throwing an InvalidPluginExecutionException with an appropriate message.
        - PreValidation steps should be executed synchronously (so the Dataverse will wait for the step to be completed).
        - Updating values is generally done in the PreOperation stage.
    - PreOperation
        - Before the main system operation, but during the database transaction.
        - You can change values for an entity included in the message here.
        - PreOperation steps should be executed synchronously.
    - MainOperation

        - For internal use only.
    - PostOperation
        - After the main system operation, but during the database transaction.
        - You can modify the message properties before the message is returned to the caller.
        - Changing entity values here will trigger a new Update event.
        - You can execute your step asynchronous or synchronous.
            - Asynchronous allows for long-running operations to be run, independent of the main Dataverse operations, improving system performance and scalability.
            - They are put in a queue, and run after the PostOperation is finished.

	PreValidation	PreOperation	PostOperation
Before main system operation	Yes	Yes	No (after)
Before database transaction	Yes	No (after)	No (after)
Asynchronous	No	No	Yes
Synchronous	Yes	Yes	Yes
Can stop record from being written	Yes	Only rollback (performance impact)	No
Can change values before record is written without triggering an Update event	Yes (but generally done in PreOperation, as the record is not created here)	Yes	No (only be triggering an Update event)

61. Design and develop a plug-in

- To create a plug-in using Visual Studio:
    - Create a new Class Library "(.NET Framework)" using .NET Framework 4.6.2.
        - Add a name for the assembly and a location.
    - In Solution Explorer, right-hand click on the project and select "Manage NuGet Packages…".
    - Install Microsoft.CrmSdk.CoreAssemblies.
        - You can also install Microsoft.CrmSdk.PluginRegistrationTool
    - You can rename the "Class1.cs" if you wish.
    - Open the class, and add:
        - using System.ServiceModel;
        - using Microsoft.Xrm.Sdk;
        - Add " : IPlugin" after the class name.
        - Click on the red underline, and VS can complete the interface.
- The plug-in must implement the IPlugin interface.
    - public class NameOfClass : IPlugin
- This exposes a single method: "Execute"

```
public void Execute (IServiceProvider serviceProvider)
```

    - This allows you to:

        - Cancel the event and display an error using the InvalidPluginExecutionException.
        - Make changes to the data.
        - Run other actions.
- Any plug-in called will allow you to get the IPluginExecutionContext interface.
    - In our C# code, we will be getting it from the serviceProvider.

```
IPluginExecutionContext context = (IPluginExecutionContext)
serviceProvider.GetService(typeof(IPluginExecutionContext));
```

    - It contains:
        - Stage – the stage in the execution pipeline,
        - ParentContext (a reference to other plug-ins which triggered this operation),
        - InputParameters – parameters that represents the operation that triggered the event. All the events except for RetrieveMultiple use the word "Target" for the relevant record.
            - var entity = (Entity)context.InputParameters["Target"];
            - RetrieveMultiple uses "Query" for the request and "Results" for the response.
            - In PreValidation and PreOperation stages, you can read and alter the InputParameters values before they are saved into the Dataverse.
- At the time of recording, you should use .NET Framework 4.6.2.
- Next you need to sign the plug-in
    - If you don't, you can't install it.
    - Right-hand click on the project in Solution Explorer, and go to Properties, then the Signing tab.
    - Click "Sign the assembly".
    - In the "Choose a strong name key file" dropbox, select "New".
    - Enter a "Key file name" (of your choice) and uncheck "Protect my key file with a password", then click OK. Then Save All and close this tab.
- Build the plug-in
    - Right-hand click on the plug-in in Solution Explorer and select "Build" to build the solution.
    - If this plug-in is built in Debug mode, it will be available in the subfolder \bin\Debug.
        - You can check this by going back to the Properties and going to the Configuration tab.
        - It will be named as per the project, with .dll afterwards.
            - To cancel the operation, throw an InvalidPluginExecutionException.
                - If synchronous, the operation will be cancelled, and an error displayed.
                - Other errors will also rollback the operation. However, InvalidPluginExecutionException is the only error which allows you to control the exception message displayed to the end user. Other errors will result is their own error being displayed, which may not be as helpful.
                - If asynchronous, an error will be logged to the System Job table (also known as the AsyncOperation Table). You can optionally ask it to try again, up to four times.

        - It is best if operations were cancelled in the PreValidation stage, so that there is nothing to rollback, as rolling back requires resources and has a performance hit.
    - OutputParameters – these represent the return value of the operation.
        - Only for use for plugins in the PostOperation stage.
    - SharedVariables – data passed either from the API or from an earlier step to a later step.
    - PreEntityImages and PostEntityImages
        - a snapshot or image of the table data. You need to have the table name surrounded by [" and "].
        - These provide much better performance than using code to retrieve a table
        - A PreEntityImage does not exist for Create, or for PostOperation stages.
        - A PostEntityImage does not exist for Delete.
        - Both images exist for Update operations.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Xrm.Sdk;
namespace Plugin
{
    public class PluginCSCode : IPlugin
    {
        public void Execute(IServiceProvider serviceProvider)
        {
            IPluginExecutionContext context = (IPluginExecutionContext)
serviceProvider.GetService(typeof(IPluginExecutionContext));
            ITracingService tracingService = (ITracingService)
serviceProvider.GetService(typeof(ITracingService));
            tracingService.Trace("Stage 1");
            if (!context.InputParameters.ContainsKey("Target"))
            {
                throw new InvalidPluginExecutionException("The target is not here");
            }
            try
            {
                var entity = (Entity)context.InputParameters["Target"];
                tracingService.Trace("Stage {0}", "2");
                if (entity.Attributes.ContainsKey("address1_line3"))
                {
                    string strAddress1Line3 = (string)entity["address1_line3"];
                    tracingService.Trace("Stage {0}", "3");
                    entity["address1_line3"] = "The data was " + strAddress1Line3;
```

```
                tracingService.Trace("Stage {0}", "4");
            }
            else
            {
                tracingService.Trace("Stage {0}", "5");
                entity["address1_line3"] = "There is no data here.";
                tracingService.Trace("Stage {0}", "6");
            }
        }
        catch
        {
            tracingService.Trace("Catch!");
            throw new InvalidPluginExecutionException("There is an error");
        }
    }
  }
}
```

62. Debug and troubleshoot a plug-in

- You can use the Tracing Service to add logs to your plug-in.
  - Firstly, you need to enable logging for Power Apps.
    - Go to Power Apps Portal – Advanced Settings – System – Administration – System settings, and go to the Customization tab.
    - Change the "Enable logging to plug-in trace log" to All.
      - All writes all the trace information, including exceptions (errors).
      - After you have finished testing your plug-in, you should either disable logging, or change it to Exception instead of All. This would capture exceptions such as the InvalidPluginExecutionException.
    - Click OK.
  - Then, in your code, you need to get an instance of the Tracing service:

```
ITracingService tracingService =
(ITracingService)serviceProvider.GetService(typeof(ITracingService));
```

  - Then you can write to the trace:

```
tracingService.Trace("This message is {0} {1}.", "over", "here");
```

    - If "Enable logging to plug-in trace log" is off, you can still use tracingService.Trace – it just won't save your message anyway.
    - To view your traces, go to Power Apps Portal – Advanced Settings – System – Customization – Plug-in Trace Log.
  - By default, a bulk deletion background job runs once a day to delete trace records more than 24 hours old.
- You can also use the Plug-in profiler to replay the execution.
  - To install the Plug-in Profiler:
    - Go to Plug-in Registration Tool and click on "Install Profiler".
    - Then click on the relevant step and select "Start Profiling" – use the default settings.
  - To replay the execution:

        - In the Plugin-Registration Tool, on the appropriate step, Click on "Debug".
        - Do the appropriate action – this will generate a log.
        - Click on the down-arrow and select the relevant profile.
        - Next to Plugin, click on the … and select your dll.
        - Note the PID next to the Plug-In Traces.
        - In Visual Studio, set breakpoints in your plug-in class.
        - Go to Debug – Attach to Process.
        - Select the relevant PID (which says "PluginRegistration.exe") and click Attach.
        - Back in the Plugin-Registration Tool, click on "Start Execution".
        - Your code will then execute, and you can set through the code, watch variables etc.
    - To stop the profiling:
        - Go to Plug-in Registration Tool and click on "Stop Profiling".

63. Implement business logic by using pre-images and post-images

- You can refer to primary table property values, including those not included in an operation.
    - For an Update operation, the execution context provides the updated value, not the value before it was changed.
- This can be done with a pre-entity image.
    - This captures a snapshot of the table with fields you are interested in before the operation.
    - Can be used for Delete and Update messages.
        - It cannot be used for Create, because there is no row before it is created.
- You can also have a post-entity image.
    - This can be used for Create and Update messages, and only of the PostOperation stage.
- To create an image.
    - In the Plug-in Registration Tool:
        - Right-hand click on the Step.
        - Click on "Register New Image".
        - Select either "Pre Image" or "Post Image".
        - Enter a Name, Entity Alias ("PreImageAlias") and Parameters (columns/fields).

```
using System;

using Microsoft.Xrm.Sdk;

namespace Plugin

{

    public class PluginCSCode : IPlugin

    {

        public void Execute(IServiceProvider serviceProvider)

        {

            IPluginExecutionContext context = (IPluginExecutionContext)
serviceProvider.GetService(typeof(IPluginExecutionContext));

            ITracingService tracingService = (ITracingService)
serviceProvider.GetService(typeof(ITracingService));

            tracingService.Trace("Stage 1");

            if (!context.InputParameters.ContainsKey("Target"))

            {

                throw new InvalidPluginExecutionException("The target is not here");

            }
```

```
            try
            {
                var entity = (Entity)context.InputParameters["Target"];

                tracingService.Trace("Stage {0}", "2");
                if (entity.Attributes.ContainsKey("fax"))
                {
                    string strNewFax = (string)entity["fax"];
                    var entityOld = (Entity)context.PreEntityImages["PreImageAlias"];
                    string strOldFax = "";
                    if (entityOld.Attributes.ContainsKey("fax"))
                    {
                        strOldFax = (string)entityOld["fax"];
                    }
                    else
                    {
                        strOldFax = "Nothing";
                    }
                    tracingService.Trace("Stage {0}", "3");
                    entity["address1_line3"] = "The data was " + strOldFax + " and is
now " + strNewFax;
                    tracingService.Trace("Stage {0}", "4");
                }
                else
                {
                    tracingService.Trace("Stage {0}", "5");
                    entity["address1_line3"] = "There is no data here.";
                    tracingService.Trace("Stage {0}", "6");
                }
            }
            catch
            {
                tracingService.Trace("Catch!");
                throw new InvalidPluginExecutionException("There is an error");
            }
        }
    }
}
```

64. Perform operations on data by using the Organization service API

- You can work with data and table and column definitions in Dataverse using the Web API, and with the Microsoft Dataverse Organization Service.
    - The Web API is an OData v4 RESTfull endpoint.
    - The Organization service uses the .NET Framework SDK.

    - They allow you to do the same things. The Web API provides a RESTful programming experience, but the data operations go through the Organization service. This makes it a superior alternative generally.
    - However, you need to use the Organization service when creating plug-ins, as plug-ins depend on the .NET Framework.
- To use the Organization service API, you need to use the IOrganizationService:

```
IOrganizationServiceFactory serviceFactory =
(IOrganizationServiceFactory)serviceProvider.GetService(typeof(IOrganizationServic
eFactory));

IOrganizationService service =
serviceFactory.CreateOrganizationService(context.UserId);
```

    - The IOrganizationService has 8 methods:
        - Execute, and Create, Retrieve, RetrieveMultiple, Update, Delete, Associate and Disassociate.
- You can use "create" as follows:

```
 var newAccount = new Entity("crbaf_accountcopy");

newAccount["crbaf_name"] = newAccountName;

newAccount["crbaf_fax"] = "765-4321";

Guid accountID = service.Create(newAccount);
```

    - You can also use CreateRequest if want to pass optional parameters.
        - This is useful if you want duplicate detection rules to be applied.
    - Instead of the last line, you can use:

```
using Microsoft.Xrm.Sdk.Messages;

 var request = new CreateRequest() { Target = newAccount } ;

            var response = (CreateResponse)service.Execute(request);

            Guid accountID = response.id;
```

- You can use "update" as follows:

```
var existingAccountCopy = new Entity("crbaf_accountcopy", accountID);

var updateAccountCopy   = new Entity("crbaf_accountcopy");

updateAccountCopy.Id = existingAccountCopy.Id;

updateAccountCopy["crbaf_fax"] = "765-4322";

service.Update(updateAccountCopy);
```

- You can use "UpsertRequest" to insert or update as necessary.
    - However, you will get less performance for using Upsert than Create.

```
UpsertRequest request = new UpsertRequest() {Target = updateAccountCopy};

UpsertResponse response = (UpsertResponse)_serviceProxy.Execute(request);
```

- You can use "retrieve" as follows:

```
var retrieveEntity = service.Retrieve("crbaf_accountcopy", accountID, new
ColumnSet("crbaf_name", "crbaf_fax"));

tracingService.Trace("Stage {0} {1}", retrieveEntity["crbaf_name"],
retrieveEntity["crbaf_fax"]);
```

- You can use "delete" as follows:

```
service.Delete("crbaf_accountcopy", accountID);
```

- You can use "retrievemultiple" as follows:

```
 ConditionExpression vCond = new ConditionExpression();

 vCond.AttributeName = "name";

 vCond.Operator = ConditionOperator.BeginsWith;
```

```
vCond.Values.Add("My Account");
// FilterExpression - contains ConditionExpression
FilterExpression vFilt = new FilterExpression();
vFilt.Conditions.Add(vCond);
// QueryExpression - ColumnSet, Table
QueryExpression vQuery = new QueryExpression("account");
// vQuery.ColumnSet.AddColumns("name");
vQuery.ColumnSet.AllColumns = true;
vQuery.Criteria.AddFilter(vFilt);
EntityCollection vResult = service.RetrieveMultiple(vQuery);
foreach (var oneResult in vResult.Entities)
{
    string b1 = "";
    if (oneResult.Attributes.Contains("name"))
    {
        b1 = (string)oneResult["name"];
    }
    string b2 = "No fax";
    if (oneResult.Attributes.Contains("fax"))
    {
        b2 = (string)oneResult["fax"];
    }
    tracingService.Trace("Stage {0} {1} {2}", ++StageNumber, b1, b2);
}
```

- Syntax notes:
  - ConditionExpression, FilterExpression and AddFilter reduces the number of rows – like the WHERE clause in SQL. ConditionOperator includes:
    - Null, NotNull
    - Numbers
      - Between, NotBetween
      - Equal, NotEqual
      - GreaterThan, GreaterEqual
      - LessThan, LessEqual
    - Strings (in addition to the "Numbers" conditions)
      - BeginsWith, DoesNotBeginWith
      - Contains, DoesNotContain
      - EndsWith, DoesNotEndWith
      - Like, NotLike
    - Dates
      - Today, Tomorrow, Yesterday
      - This/Last, On, In
  - QueryExpression starts off by naming the table – like the FROM clause in SQL
  - ColumnSet shows what columns to retrieve – like the SELECT clause in SQL.

65. Optimize plug-in performance by configuring concurrency, transactions, and batching

- You can view analytics by:
  - Going to Power Apps Portal – Admin center – Analytics – Dataverse.
  - The Plug-ins tab has:
    - Plug-in Success Rate
      - The pass rate divided by total plug-executions
    - Plug-in Executions
      - The number of plug-ins which have been executed
    - Most active plug-ins,
      - Top 10 most executed plug-ins.
    - Average plug-in execution time
      - Average time to execute a plug-in
    - Top plug-ins by failures
      - Top 10 most failing plug-ins.
- Concurrency – An alternative to Update is UpdateResponse.
  - Instead of
    ```
    service.Update(account);
    ```
  - you can use
    ```
    var request = new UpdateRequest()
    { Target = account };
    service.Execute(request);
    ```
- UpdateResponse (or DeleteResponse) should be used if:
  - You want duplicate detection rules to be applied for an Update, or
  - The data that you have may be stale (it has since been changed), and you want to cancel the update or delete if it is no longer relevant.
    - It uses the RowVersion of the database, and compares it with the currently stored row
  - You want to specify an optimistic concurrency behavior.
- Concurrency happens when two update/delete operations happen on the same piece of data at the same time.
    - All custom tables support this by default.
    - There are 3 ConcurrencyBehavior values:
      - 0 = Default – let the computer decide.
      - 1 = IfRowVersionMatches – only if the target data has the same row version. If not, fail the update/delete.
      - 2 = AlwaysOverwrite – always update or delete.
    - You can specify:
      ```
      var retrievedAccount = new Account()
      {
          Id = new Guid("a976763a-ba1c-e811-a954-000d3af451d6"),
          RowVersion = "986323"
      };
      var account = new Account();
      account.Id = retrievedAccount.Id;
      account.RowVersion = retrievedAccount.RowVersion;
      account.CreditOnHold = true;
      ```

```
var request = new UpdateRequest()
{Target = account,
    ConcurrencyBehavior = ConcurrencyBehavior.IfRowVersionMatches
};
service.Execute(request);
```

- In order to do an operation, a transaction is created.
    - This transaction allows for rollback (undone) if it fails.
    - Once the operation succeeds, the changes are committed (saved) and the transaction ends.
        - However, if one operation triggers another, then the transaction will continue.
    - During a transaction, the relevant record (or page, or table) may be locked for reading and writing.
        - This prevents "dirty reads" – reading data that has not been committed and may be rolled back.
- How does one operation impact on others?
    - Retrieve
        - Read lock on one record.
        - Low impact on other records.
    - RetrieveMultiple
        - Potentially may block multiple records or pages.
    - Create
        - Creating a new record so, as no-one will be using that record anyway, there are no blocking issues!
        - Low impact on other activities. However:
            - Might block locking queries to the whole table until new record is created, and
            - May trigger other customized actions which may impact other operations.
    - Update
        - Will block any other operation updating or reading that record.
        - Will block table-level operations
        - May trigger other activities.
- Therefore, with your transactions, make them as small as possible.
    - Pre-validation and pre-operation transactions should ideally be limited to short-term validation of input values.
    - Post-operation synchronous actions should also be short running. They can be rolled back in the event of failure.
    - If you have a medium length action which does not need to be rolled back in the event of failure, then that should be an asynchronous transaction, as that does not affect the rest of the pipelines.
        - Ideally, don't have very low cost actions as asynchronous, as there is additional overhead with asynchronous processing.
- You can also Batch your transactions.
    - This is useful if you need to the entire transaction to fail, if one part of it fails.
- Batching is asking for multiple requests at the same time.
    - It may degrade user experience if done synchronous.
    - It could also timeout after 2 minutes.

    - Operations in a batch request are done sequentially.
        - You are losing the possibility of them being done using parallel processing, thereby reducing overall time.
- See this web page for an example of a batched transaction using ExecuteTransactionRequest.
    - For external clients which require the operations to be batched as a single transaction.
    - It is included in Github Code samples \dataverse\orgsvc\C#\ExecuteMultipleInTransaction
    - This create a new ExecuteTransactionRequest, with
        - Requests – an array containing individual requests, and
        - ReturnResponses.
    - It then creates multiple entities in memory.
    - Then adds into the ExecuteTransactionRequest these individual CreateRequest.
    - It then executes all of these requests using Execute.
    - And then loops through each of the responses.
- See this web page for an example of an example using ExecuteMultipleRequest.
    - Only use this to bulk load data,
    - deliberately for long-running external processes, or
    - to reduce the round trips between custom client and Dynamics 365 servers, reducing the total latency.
    - It is included in Github Code samples\cds\orgsvc\C#\ExecuteMultipleInTransaction
    - This allows for ContinueOnError (yes/no) as well as ReturnResponses

66. Configure a Dataverse custom API message

- "Message" is in the sense of Create, Retrieve, RetrieveMultiple, Update, Delete, Associate or Disassociate.
    - These are built-in messages.
    - However, you can create a custom message for your own events.
- You can create your messages:
    - Using the plug-in registration tool
    - In the Power Apps Portal,
    - With code, or
    - With solution files.
- To create a custom API using the Plug-in Registration Tool:
    - Go to Register – Register new Custom API command.
    - Note: you don't need to set any step – just register the API command.
    - This new form includes:
        - Display Name – the localizable name.
        - Name – a friendly, non-localizable name.
        - Solution – you can select a new or an existing solution.
        - Unique Name – for the Custom API.
            - Alphanumeric characters only – no spaces.
            - This cannot be later changed.
        - Description – localizable description.
            - Used when exposed in an app.
        - Assembly and plugin.
            - Optional. The assembly and plug-in type.
        - Allowed Custom Processing Step Type. This is not alterable later. Either
            - Sync and Async

            - This is the standard Step Type. You should use this only you have reasons to change it. This step is customizable.
            - Other developers can detect this message, change its behavior and cancel it.
            - They can detect the message and add logic to run asynchronously.
        - None
            - Other developers cannot customizable this message.
            - They cannot trigger additional steps to this event, modify its behavior, or cancel it.
        - Async only
            - Other developers can detect this message, but cannot cancel it or customize its behavior.
    - Binding Type. This is not alterable later.
        - Global – operation does not apply to a specific table.
        - Entity – operation returns a single record of a specific table. Returns as a parameter called "Target".
        - EntityCollection – Changes or returns a collection of a specific table.
    - If you use Entity or EntityCollection:
        - You need to enter the "Bound Entity Logical Name", which contains the table's logical name. This is not alterable later.
        - When using the Web API, you will need to use the fully qualified name of the action, which starts with "Microsoft.Dynamics.CRM.".
    - Execute Privilege Name.
        - Someone who will control whether another person can use the API.
    - Function? (True/False)
        - A function uses a GET request, so it retrieves data without changing any data.
            - The parameters are passed in the URL.
            - The URL has a limit of around 2,000 characters.
            - There must be some Response property.
        - An action uses a POST request.
            - If the operation is to be performed using Power Automate, you should use an Action.
    - Private? (True/False)
        - "Private" indicates you do not support other developers using your message.
        - If your API message is private, then it will not be exposed in the $metadata service document.
        - However, if other developers know about the message, they can use it.
- You can also create request or request parameters by clicking on "+ Add Request Parameter" or "+Add Response Parameter".
- A Parameter includes:
    - Display name – localizable.
    - Name – Primary name. It is recommend that this is a Unique name, by adding the Custom API name and a dot before it.
    - Unique Name – name of the parameter when calling the Custom API. This is not alterable later.

    - Type – either Boolean, DateTime, Decimal, Entity, EntityCollection, EntityReference, Float, Integer, Money, Picklist, String, StringArray or Guid.
    - Logical Entity Name – the table, if you have selected Entity, EntityCollection, EntityReference as the Type.
    - Description – localizable.
    - For Request parameters: Is Optional – whether this parameter is required. This is not alterable later.
    - Don't forget – a function must have at least 1 Request parameter. An action can have 0 or more.
- You can subsequently click the update or delete parameter icons, if needed.
- You can verify that it is installed by:
    - Going to the Power Apps Portal.
    - Clicking on Advanced Settings.
    - In the URL, delete everything after ".com", and add " /api/data/v9.1/$metadata".
    - Search for your custom API name.
- You can also create a Custom API record in the Power Apps portal.
    - By going to Solutions – [Relevant solution] – New – More – Other – Custom API.
    - You can create Custom API Request and Response parameters by going to New – More – Other – Custom API Request/Response Parameter.
        - You will need to include the relevant Custom API in the form.
    - However, Microsoft says that the Plug-in registration tool is a better experience.

67. Register custom assemblies by using the Plug-in Registration Tool

- To install the Plug-in Registration Tool
    - In Visual Studio, right-hand click on References and select “Manage NuGet Packages”.
    - Search for and install Microsoft.crmsdk.XrmTooling.PluginRegistrationTool.
    - Open Windows Explorer, and go to the project and then packages/Microsoft.Crmsdk.XrmTooling.PluginRegistrationTool/tools/PluginRegistration.exe
        - If it is not there, then it may have been in a different project.
- You need to have System Customizer or System Administrator role to register a plug-in..
- To connect to the Plug-in Registration Tool.
    - Open PlugInRegistration.exe
    - Click "Create new Connection".
    - Select "Office 365".
    - Either:
        - Sign-in as current user, or
        - Click "Show Advanced" and then enter credentials.
        - You may also wish to check "Display list of available organizations", if you have multiple environments.
    - Click "Login" and, if necessary, select your environment/organization.
- You will then see a list of, among other items, existing plug-ins.
- To register the assembly:
    - Go to Register – Register New Assembly.
    - Click the … next to Step 1, and select the assembly (.dll) file.
    - Leave everything else as is.
    - Click "Register Selected Plugins".
    - You will see a dialog box showing that 1 Assembly and 1 Plugin have been registered.

    - Click OK and Close.
    - You will see the new Assembly, which contains a (Plugin).
        - You will see them at the top or bottom of the assemblies.
- You then need to register a step, which connects to the event pipeline.
    - Right-hand click on the relevant "(Plugin)" and select "Register New Step".
        - The "(Plugin)" is inside the "(Assembly)".
    - Select:
        - Message: Create, Retrieve, RetrieveMultiple, Update, Delete, Associate or Disassociate
        - Primary Entity – this is the table.
        - Secondary Entity is not typically used any more.
    - Event Pipeline Stage of Execution – PreValidation, PreOperation or PostOperation
    - Execution Mode – Synchronous or Asynchronous.
    - Click "Register New Step".
    - This step will now be shown in the "(Plugin)".
- You can now test this plug-in in Power Apps.
    - You can view the System Jobs by going to Power Apps – Advanced settings – System – System Jobs, and change the Entity filter to the relevant table.

68. Develop a plug-in that targets a custom action message

- To access the request parameters, use:
```
string requestParameter = (string)context.InputParameters["RequestParameter"];
```
- To access to response parameters, use:
```
context.OutputParameters["ResponseParameter"] = responseParameter;
```
- To call the API, then use:
```
var request = new OrganizationRequest("pca_myCustomAPI") { ["RequestParameter"] = "hi
there"};

var response = service.Execute(request);

tracingService.Trace((string)response["ResponseParameter"]);
```
- Interfaces
- IOrganizationService
    - retrieve records from Dataverse, using create, update and upsertrequest, retrieve and retrievemultiple, and delete.
- IServiceProvider
    - used in the Execute statement to retrieve an object which provides custom support to other objects.
- ITracingService
    - connect to the trace log.
- IPluginExecutionContent
    - links to the run-time environment, execution pipeline, and entity business information.
- IPlugin
    - used as the class
```
public class NameOfClass : IPlugin

{

public void Execute (IServiceProvider serviceProvider)

{
```

Create custom connectors

69. Create a definition for the API

- Microsoft Power Automate and Power Apps over several hundred connectors to Microsoft and other services.
- However, you may wish to connect to other services. This uses a custom connector.
- To create a new connector:
    - In the Power Apps portal, go to Dataverse – Custom connectors.
    - Click "+ New custom connector" – "Create from blank".
    - Enter the custom connector name.
    - On the General tab:
        - Enter a description.
        - Enter the host (the root name – e.g. pl400func.azurewebsites.net) and the Base URL (e.g. "/api").
        - You can also add an icon (a small PNG or JPG) and a background color.
    - Enter a user-friendly Label, name and location.
    - Keep the Authentication type as “Basic authentication”.
    - Click "Create connector".
    - Click on "+ New action".
        - Enter a summary and description, together with the Operation ID (a unique string used to identify the operation)
    - The visibility for parameters and operations can be:
        - none – displayed normally in the logic app or flow,
        - advanced – hidden under another menu
        - internal – hidden from the user, or
        - important – always shown to the user first.
    - In "Request", click "Import from sample".
        - In the verb, select POST.
        - In the URL, add the URL for your source (such as an Azure function),
        - You can optionally in Body enter a fully formed JSON
    - In "Response", you can optionally click on "Add default response".
        - Enter the Response body, and click Import.
    - Check the Validation area, then click "Update connector".
    - Next to request, click body - Edit.
        - Click each parameter in turn, and check:
            - Title, Description, Default Value, Is Required, and Type.
- To test the connection:
    - Go to Test – New connection.
    - Enter the API key if applicable, and click "Create connection".
        - In Power Automate, in the Test tab, click the Refresh icon.
        - In Power Apps, go to Dataverse – Custom connectors – [Select connector] – Test tab.
    - Enter relevant fields and click "Test operation".
    - You can then see the Request and Response.
- An alternative way of creating the definition is by using an API definition.
    - You can get this from the Function app settings – API – API definition, and clicking on "Generate API definition template".

        - The definition will be in OpenAPI 2.0 format, which is otherwise known as Swagger format.
- To import it:
    - In the Power Apps portal, go to Dataverse – Custom connectors.
    - Click "+ New custom connector" – "Import an OpenAPI file".
- To use it in a Automate flow:
    - Go to New step – Add an action.
    - Search for the connector, then select an action.
    - Enter values.
    - Use any output values from that step in future steps.
- To use it in a Power Apps canvas app:
    - Go to "Connect to data", and in the Data panel, choose your connection.
    - In a property (such as an OnSelect property), type the name of the connector, then a dot.
    - Then enter the action name.
    - Then enter the parameters in ({ })
    - Then add a dot, and the output
      `ConnectorName.ActionName({Para1: 1, Para2: "value"}).outputparameter`

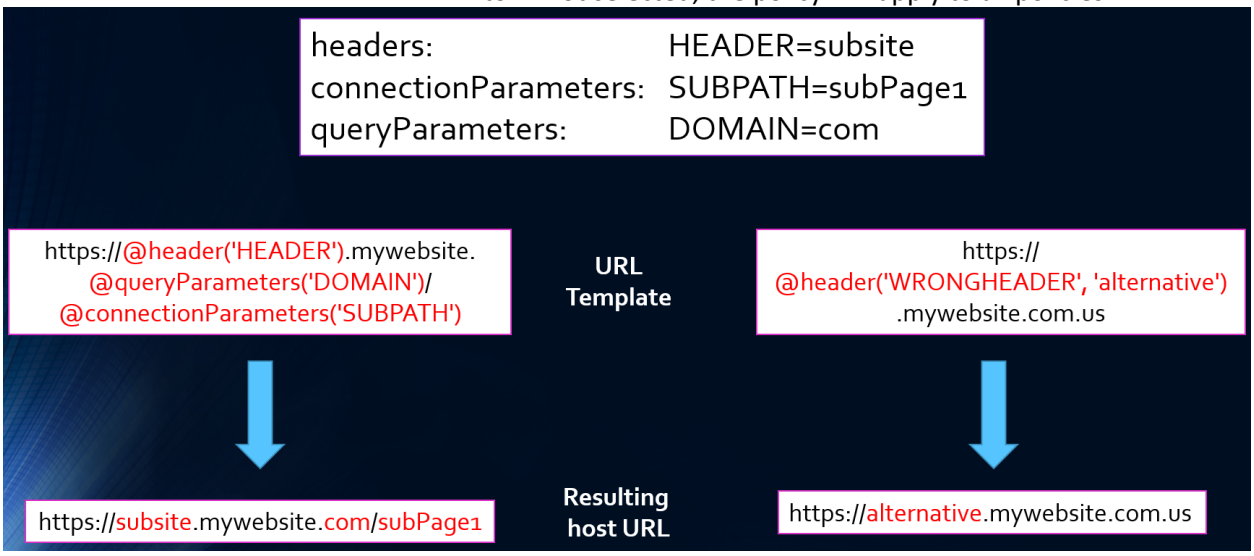
70. Configure API security

- Authentication type. The OpenAPI (Swagger) authentication types are:
    - Basic authentication
        - Built into HTTP, uses username and password. Should be used in conjunction with HTTPS or SSL.
    - API key (header or query string parameter)
        - Uses a special token (secret) which only the client and server know. The server generates this unique key.
    - OAuth2 common flows
        - Uses access tokens, allows users to share content without sharing credentials. Flows (or "grant types") are use cases, most notably "authorization codes". Used by Facebook, Google, Twitter, Microsoft and LinkedIn.

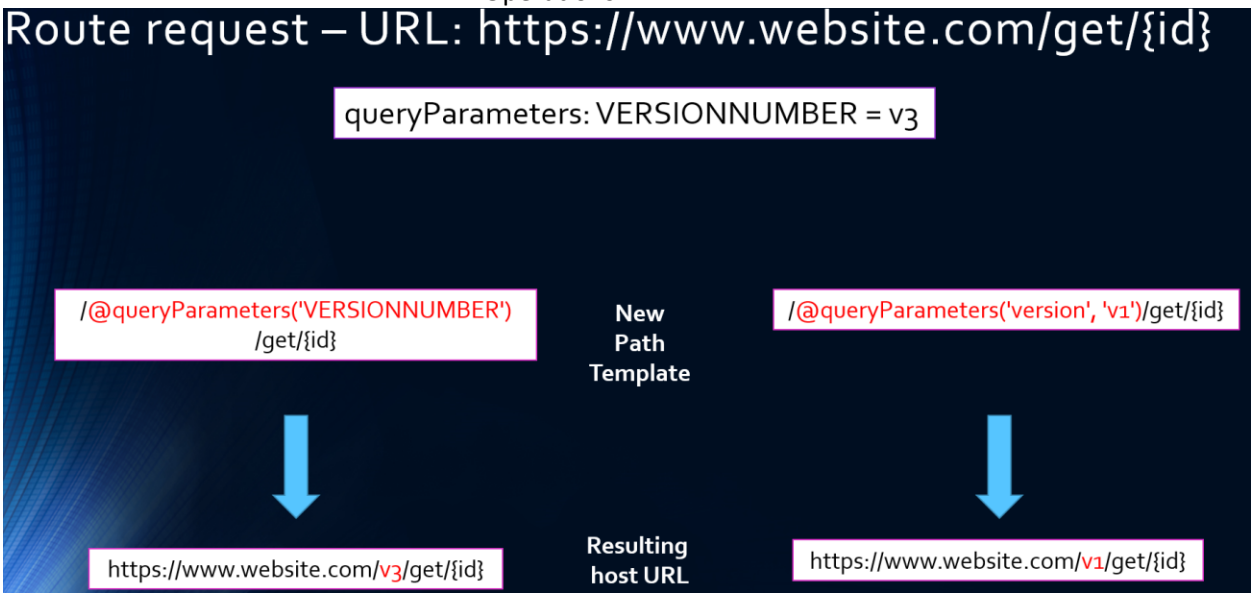
71. Use policy templates to modify connector behavior at runtime

- Policies can modify connector behavior at runtime.
    - Set host URL or Route request,
    - Set HTTP header or query string parameter.
- To use a policy template:
    - Go to the Power Apps/Automate Portal – Dataverse – Custom connectors.
    - Create a new or edit an existing custom connector.
    - Go to the Definition page.
    - On the left-hand side, go to Policies, click "+New policy".
    - Enter a policy name, and choose a template:

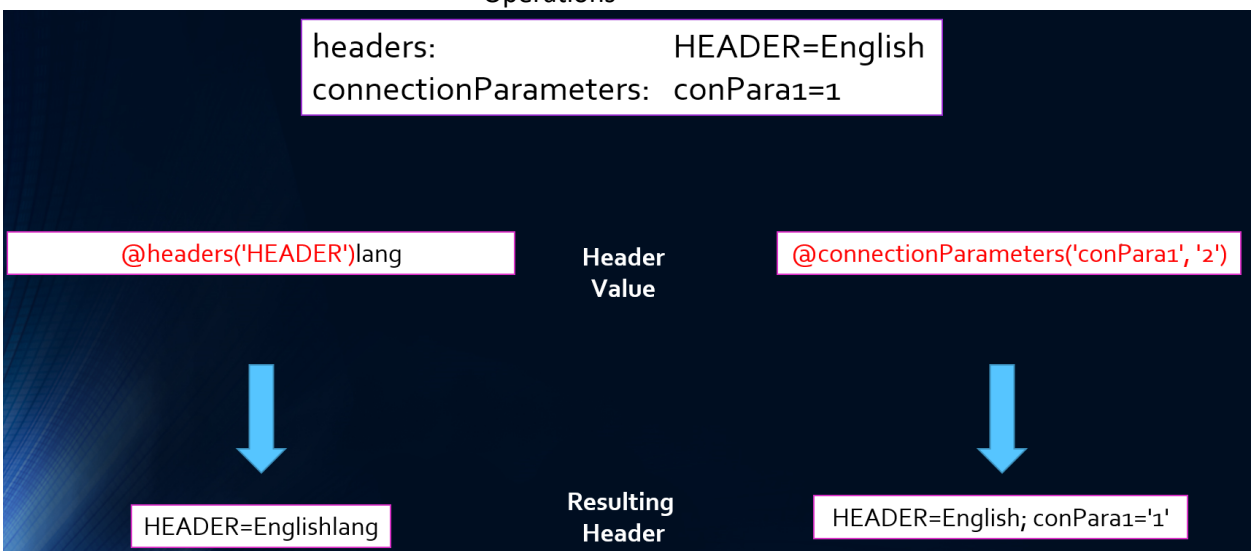
- Set host URL
  - Replaces the host URL with a URL from the template. Parameters:
    - URL template – the template to generate the host URL. You can use the expressions @headers, @queryParameters and @connectionParameters
    - Operations – the actions and triggers that the policy will apply to. If not selected, the policy will apply to all policies.



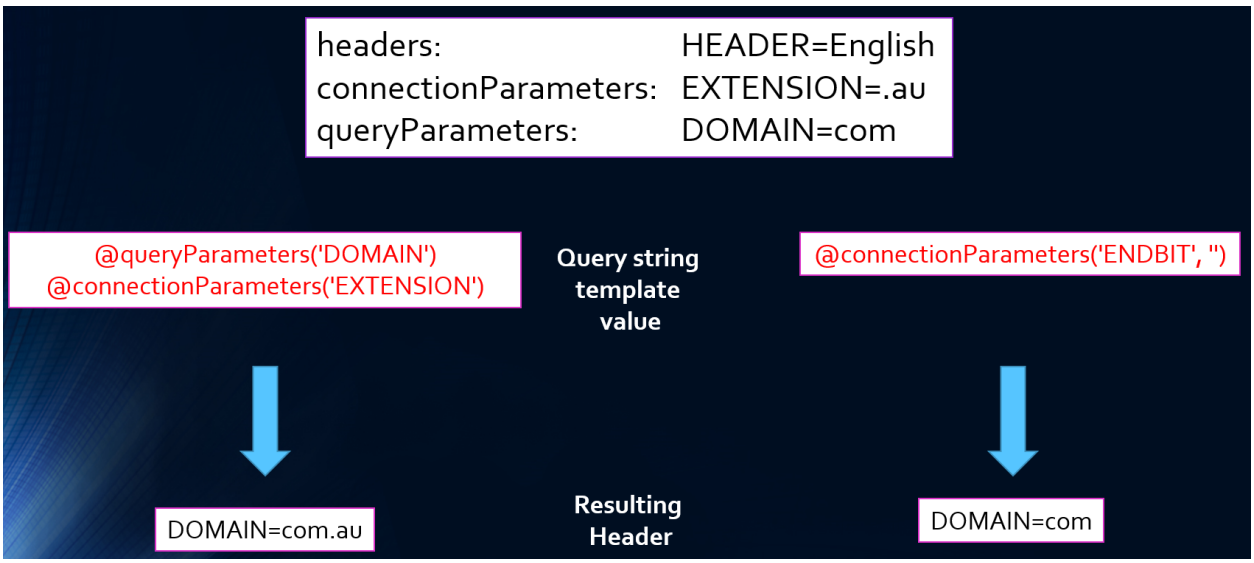
- Route request
  - Routes incoming requests to a specific endpoint on the same service. Parameters:
    - New path – relative path (you can use expressions),
    - Method – the HTTP method to use (by default – the original HTTP method),
    - Operations.



- Set HTTP header
  - Assigns a value to an existing response and/or request header.
  - Adds a new response and/or request header.
  - Parameters:
    - Header name/value – the name/value to be set.
    - Action if header exists – override, skip (if there is an existing value), append (add at end),
    - Run policy on – Request, Response, Failure (if the backend API fails),
    - Operations



- Set query string parameter.
  - Query string parameters are set from the Definition – Actions - Request
  - Adds or changes the request query string parameter.
  - Parameters:
    - Query parameter name/value – the query parameter name/value to be set,
    - Action if query parameter exists – override, skip, append
    - Operations
    - Set host URL



72. Create custom connectors for public APIs by using Postman

- Postman is an app for making HTTP requests.
  - Download it from www.postman.com
- Postman collections organize and group related API requests.
  - Click "+Create Collection" to create a collection name.
- To create an HTTP request for the API
  - Create a new request.
  - In the Authorization tab, enter the:
    - HTTP method (e.g. POST),
    - Request URL, and
    - Authorization.
  - In the Headers tab, enter the request headers.
  - In the Body tab, enter what you want to send in the request body (you could use the "raw" option).
- You can then save the request response.
  - Click on "Save Response" (on the right-hand side).
  - Provide a name for the example response.
- You can then export the Postman collection
  - You need to remove the headers by clicking on the X next to them.
    - These headers were needed for API requests, but are handled differently in the custom connector.
  - Go to the collection, click the … and click Export.

- Choose the Collection format.
  - Power Platform used to need version 1.0, but it can now use versions 2.0 and 2.1 as well. You can now create a custom connector
- Go to Power Apps/Automate Portal – Dataverse – Custom connectors
- Click "+ New custom connector" – "Import a Postman collection".
- Fill in any remaining details. You may wish to edit the parameters to make them more user-friendly.

Use platform APIs

73. Interact with data and processes by using the Dataverse Web API or the Organization Service

- See topics 50 and 64

74. Implement API limit retry policies

- API limits are there to stop client applications from taking too many server resources.
  - Only used for very high request levels.
- When the limits are reached, then you will receive:
  - For Web API, the 429 Too Many Requests error
  - For the Organization Service, the OrganizationServiceFault error – either:
    - Number of requests exceeded the limit of 6000 over a time window of 300 seconds.
    - Combined execution time of incoming requests exceeded limit of 1,200 seconds over time window of 300 seconds.
    - Number of concurrent requests exceeded the limit of 52.
- If you get the Web API error, then you can add a delay and retry it.
  - For an example, see Github Code samples\cds\webapi\C#\CDSWebApiService.cs, lines 460-511
- If you are using the Organization service, then use "Microsoft.Xrm.Tooling.Connector.CrmServiceClient".
  - This implements the IOrganizationService methods, and can handle any API limit errors.
  - Since version 9.0.2.16, it will automatically pause and re-send the request after the Retry-After duration period.

```
private async Task<HttpResponseMessage> SendAsync(

HttpRequestMessage request,

HttpCompletionOption httpCompletionOption = HttpCompletionOption.ResponseHeadersRead,
int retryCount = 0)

{

HttpResponseMessage response;

response = await httpClient.SendAsync(request.Clone(), httpCompletionOption);

if (!response.IsSuccessStatusCode)

{

        if ((int)response.StatusCode != 429)

            { throw ParseError(response);  }

        else

        {

            if (++retryCount >= config.MaxRetries)
```

```
            { throw ParseError(response); }
await Task.Delay(TimeSpan.FromSeconds(2));
return await SendAsync(request, httpCompletionOption, retryCount);
        }
    }
    else
    {
        return response;
    }
}
int seconds;
if (response.Headers.Contains("Retry-After"))
{
seconds = int.Parse(response.Headers.GetValues("Retry-After").FirstOrDefault());
}
else
{
//Otherwise, use an exponential backoff strategy
seconds = (int)Math.Pow(2, retryCount);
}
await Task.Delay(TimeSpan.FromSeconds(seconds));
return await SendAsync(request, httpCompletionOption, retryCount);
```

75. Optimize for performance, concurrency, transactions, and batching

- See topic 65.

76. Perform authentication by using OAuth

- For an example of how to use OAuth in your .Net programs, see Github Code samples\cds\webapi\C#\QuickStart\QuickStartMSAL
    - MSAL is the Microsoft Authentication Library.
        - There is also the ADAL (Active Directory Authentication Library). However, it is no longer supported (support ceased in June 2022 and is no longer being updated), so you should not use.
    - Add using Microsoft.Identity.Client to add the relevant library.
    - PublicClientApplicationBuilder.Create(clientId).Build(); initializes an application from code
        - The ClientID is the unique app assigned for your app by Azure AD.
        - .WithAuthority() sets the default authority to an AAD authority. The Authority is the identity provider instance, the sign-in audience, and potentially the tenant ID
        - .WithRedirectUri overrides the default redirect URI.
            - This is the URI the security tokens will be sent to from the identity provider.
            - This is shown in "Register an application" – Redirect URI.
    - The "scope" is your Dataverse.

        - It can be found by going to the Power Apps – [Choose the relevant environment] –Settings (wheel) – Developer resources.
        - Use the "Web API endpoint" starting with https:// and ending with ".com" (so do not include the "/api/data/v9.2").
    - You then need to request an access token.

Graph API

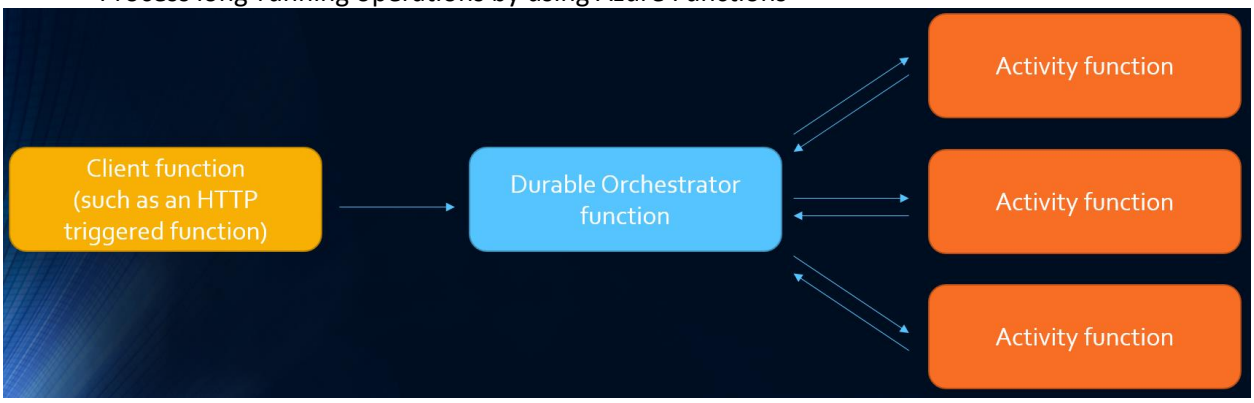
- Microsoft's Graph API allows you to connect to:
    - Microsoft 365
        - Bookings (appointment scheduling)
        - Calendar and To Do
        - Excel workbooks and charts, OneNote
        - Microsoft 365 compliance eDiscovery
        - Microsoft Search
        - OneDrive and SharePoint files
        - Outlook/Exchange, including Outlook calendar, contacts and mail
        - Planner tasks and plans
        - Teams for calls, online meetings, messaging, and sharing files, notes, calendar and plans.
        - Users and groups
    - Services in Windows 10 and above
        - Devices, such as device updates and Cloud PC (Virtual Machines)
        - Microsoft 365 Lighthouse (managing multiple customer organizations using Microsoft 365 Business Premium)
        - Universal Print (cloud printing)
    - Enterprise Mobility and Security
        - Advanced Threat Analytics and Protection
        - Azure Security Center
        - Microsoft Defender for Cloud Apps
        - Windows Defender Advanced Threat Protection.
        - Identity Manager
        - Intune (Corporate management of devices and apps)
    - Identity and Access Management
        - Azure Active Directory
    - You can connect from data sources such as:
        - Box
        - Google Drive
        - Jira
        - Salesforce.
- There is a single Microsoft Graph API endpoint, https://graph.microsoft.com
    - You can use a custom connector and use it in (for example) an Azure Function.
- Microsoft Graph Data Connect gives you tools to connect to Graph data at scale using Azure Data Factory (not RESTful web queries).

Process workloads

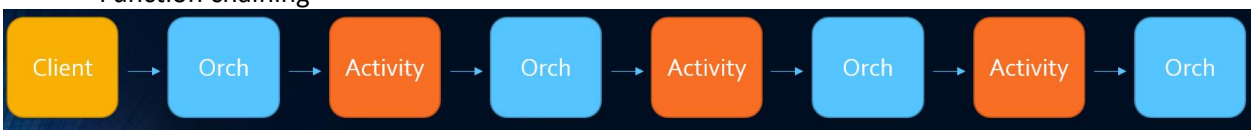
77. Process long-running operations by using Azure Functions

- Azure Functions are limited to:

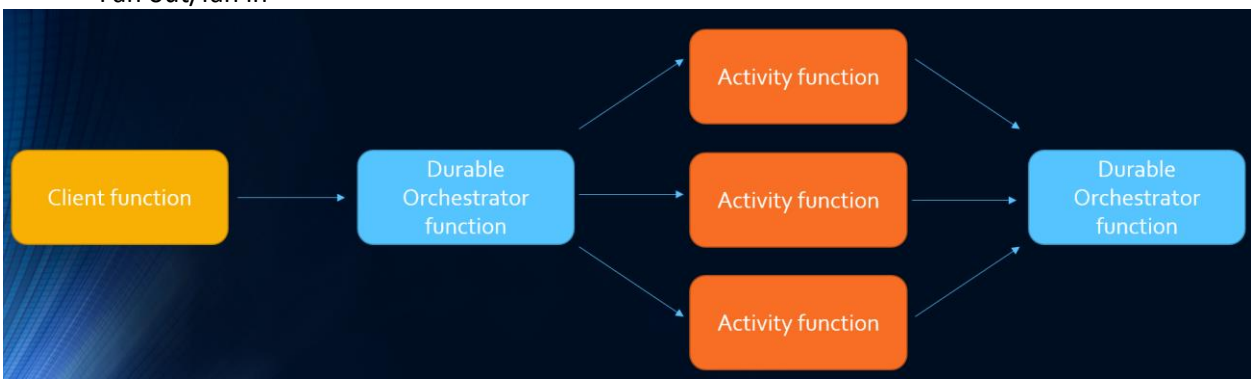
    - 10 minutes for a single execution in the Consumption plan, and
    - 30 minutes for the Premium plan.
- To create a long-running operation, you need a Durable Function.
    - You can use .NET, C#, F#, Javascript/TypeScript, Python, PowerShell and Java (in preview).
    - Durable functions allow functions to be chained to each other.
- There are:
    - Activity functions
        - The work is performed in activity functions.
        - The results from these other functions can be saved as variables.
    - A durable orchestrator function.
        - This define function workflows using code, calling activity functions either synchronously or asynchronously.
        - Progress is stopped until the function "await" (in C#) or "yield" (JavaScript).
        - These functions can last a very short time (seconds) to a very long time (years).
    - A client function
        - An Azure Function that starts an orchestrator function.
- Process long-running operations by using Azure Functions



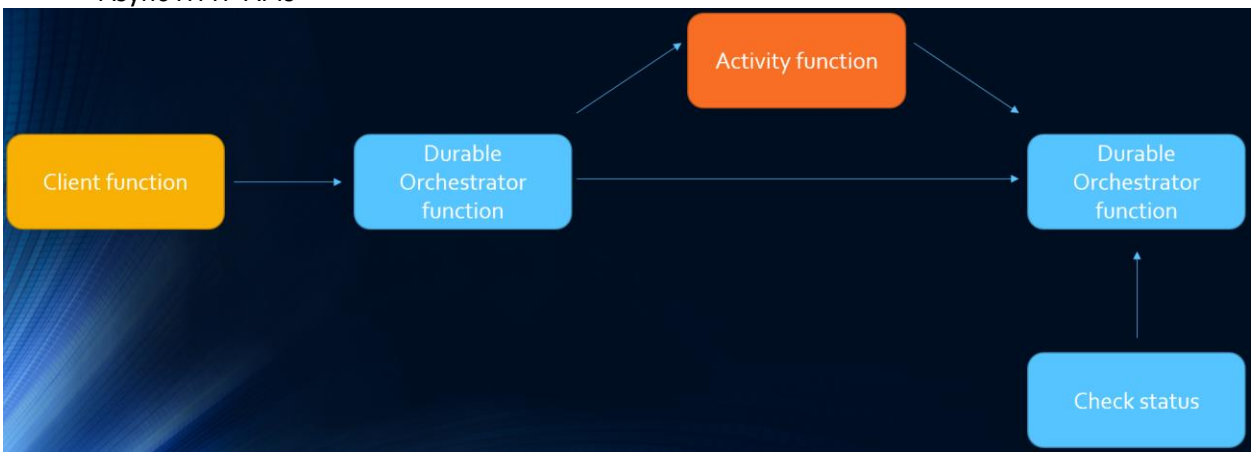
- Function chaining



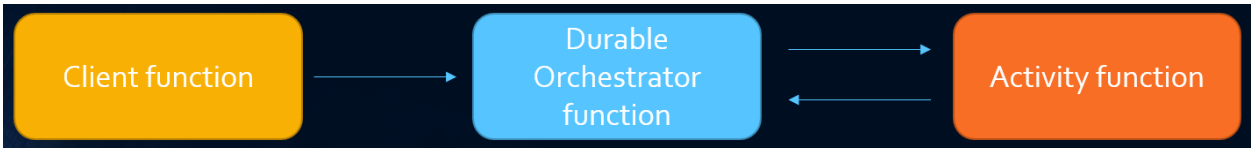
- Fan out/fan in



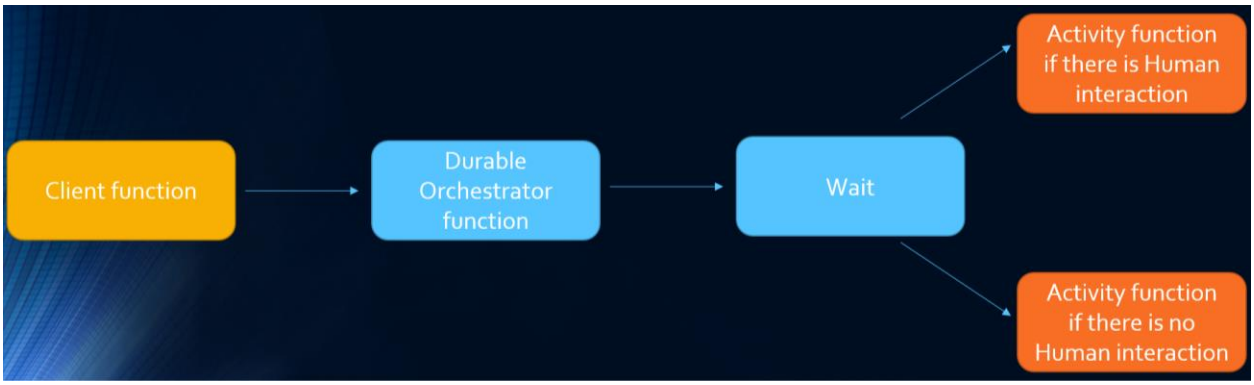
- Async HTTP APIs



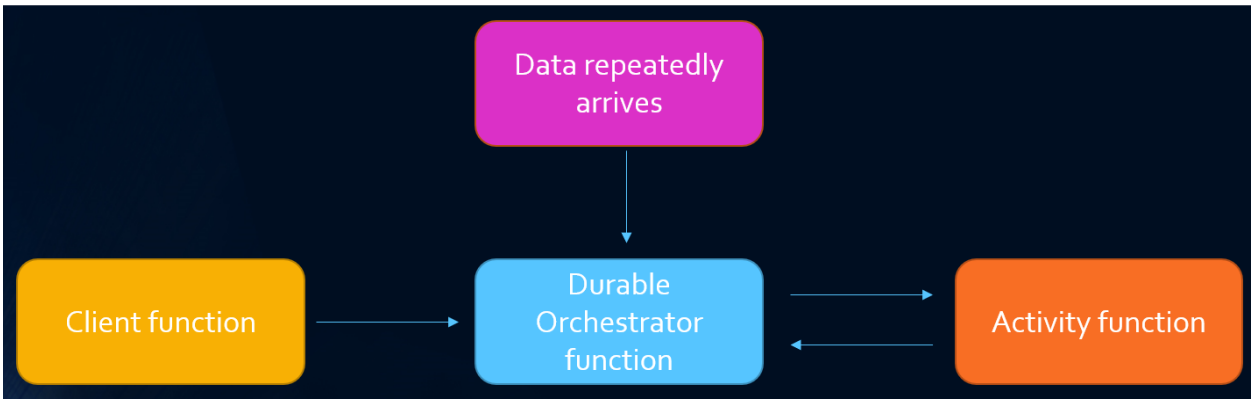
- Monitor



- Human interaction



- Aggregator



- To create a long-running operation:
  - Create a Function App

    - Choose the Runtime stack:
        - Choose between .NET, Node.js, Python, Java, PowerShell Core and Custom Handler. (In this example, choose Node.js).
        - Change the Version and Region as appropriate.
        - The plan type should be "Consumption (Serverless)" for this test function.
        - Leave every option as is.
        - Go to the "Review + Create" tab and click Create.
        - Click "Create resource".
- Go to Development Tools – App Service Editor:
    - Click on "Go ->", then click on wwwroot
    - Go to Console (on left-hand side), and touch package.json
    - Go to "package.json", and enter:
        - { "name": "example",
        - "version": "1.0.0" }
    - Make sure it is saved.
- Go to Development Tools – Console. Enter:
    - npm install durable-functions
- This required the "package.json" file. Wait until it has installed – it will take a couple of minutes.
- Go to Overview and click Restart.
- Create a Durable Functions HTTP starter.
    - Go to Functions – Functions.
    - Create "+ Create".
    - Note: you can choose the Development environment from "Develop in portal", "VS Code" and "Any editor + Core Tools".
    - Click on "Durable Functions HTTP starter".
    - Enter a Function name.
    - Click "Create".
    - Look at the code by clicking on "Code + Test".
    - Click on "Get function URL", and click on Copy next to URL. Paste it into NotePad.
- The default code is for Node.js:

```
const df = require("durable-functions");

module.exports = async function (context, req) {

    const client = df.getClient(context);

    const instanceId = await client.startNew(req.params.functionName, undefined,
req.body);

    context.log(`Started orchestration with ID = '${instanceId}'.`);

    return client.createCheckStatusResponse(context.bindingData.req, instanceId);

};
```

- Create a "Durable Functions Orchestrator" function – click on the Function App name in the breadcrumbs at the top, and create a new function.
    - Create it, and look the code in "Code + Test".
    - This code sends to an activity function called "Hello".
    - The default Node.js code is:

```
const df = require("durable-functions");
```

```
module.exports = df.orchestrator(function* (context) {
    const outputs = [];
    // Replace "Hello" with the name of your Durable Activity Function.
    outputs.push(yield context.df.callActivity("Hello", "Tokyo"));
    outputs.push(yield context.df.callActivity("Hello", "Seattle"));
    outputs.push(yield context.df.callActivity("Hello", "London"));
    // returns ["Hello Tokyo!", "Hello Seattle!", "Hello London!"]
    return outputs;
});
```

- Create a "Durable Functions activity" function called "Hello".
- The default Node.js code is:

```
module.exports = async function (context) {
    return `Hello ${context.bindings.name}!`;
};
```

- In NotePad, change "{functionName}" to your Orchestrator function name.
- Paste that URL into your web browser.
- Then copy the "statusQueryGetURi" string into your web browser.

78. Configure scheduled and event-driven function triggers in Azure Functions

- Azure Functions is a form of serverless computing.
  - You write the code and publish it to Azure.
  - You don't need to worry about scaling, operating and resilience.
  - The cost is also good.
    - On the Consumption tier, the first 400 MB/s of execution and 1,000,000 executions per month are free.
    - If your code doesn't run, you don't pay anything.
- Languages:
  - C# script, JavaScript and PowerShell can be run in Linux and Windows, and can also be edited in the Azure portal.
  - Python can be run in Linux and can be edited in the Azure portal. You must develop your functions locally.
  - C# class libraries, Java, TypeScript and Go and Rust can be run in Linux and Windows, but cannot be edited in the Azure portal.
- Visual Studio 2019 onwards includes Azure Functions tools for C# class library and script.
  - If you select the Azure development work in your VS installation.
  - You can also install it in Visual Studio 2017.
  - You can then go to File – New – Project.
  - In the Create a new project, choose the "Azure Functions template".
  - Enter a project name (alphanumeric characters only).
  - In Additional information, select:
    - the Functions worker of .Net 6,
    - The Function template of the HTTP trigger.
    - A Storage Account will be assigned or created when you publish your project to Azure.
  - Click Create.
    - It creates a C# project, installs the Microsoft.NET.Sdk.Functions, and sets the target framework.

- The file host.json allows you to configure the Functions host, and local.settings.json are local settings only.
    - As local.settings.json may contain secrets (for example, connection strings), it should never be stored remotely. This file is not uploaded when the project is published.
- The default code for C# is:

```
#r "Newtonsoft.Json"

using System.Net;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Primitives;

using Newtonsoft.Json;

public static async Task<IActionResult> Run(HttpRequest req, ILogger log)

{

    log.LogInformation("C# HTTP trigger function processed a request.");

    string name = req.Query["name"];

    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();

    dynamic data = JsonConvert.DeserializeObject(requestBody);

    name = name ?? data?.name;

    string responseMessage = string.IsNullOrEmpty(name)

      ? "This HTTP triggered function executed successfully. Pass a name in
the query string or in the request body for a personalized response."

      : $"Hello, {name}. This HTTP triggered function executed successfully.";

            return new OkObjectResult(responseMessage);

}
```

- Add a new Azure Function, if you want a second function.
    - Right-hand click your project in Solution Explorer and go to Add – New Azure Function.
    - Add a class name, and click Add.
    - Choose the trigger – for example, Http trigger.
        - If you are using a Timer trigger, then you will need to enter the schedule in CRON date time format.
            - This consists of 6 values: seconds, minutes, hours, day, month (1-12 or JAN-DEC), and day of week (where 0 or 7 = Sunday).
            - A * shows all the values – for example, * in the hours is every hour.
            - A hyphen shows a range – for example, 0-23 is every hour.
            - A / shows an interval – for example, 0-23/2 every other hour.
            - A , shows a range – for example, 1,2,13,14 shows 1 o'clock and 2 o'clock in the morning and afternoon.
    - Select a provisioned Azure storage account, or a storage emulator.
    - Note: a function must have one trigger (no more, no less). But different functions can have different triggers.
- You can run functions locally by pressing F5 or Play.
    - You may get a request to download and install Azure Function Core (CLI) tools.
    - You may also need to enable a firewall exception so that it can handle HTTP requests.

    - It runs the local Functions host (func.exe), which starts to listen a local port (typically 7071).
    - You can then test your code. You can use breakpoints when testing.
- You can then publish your Function.
    - Right-hand click the project in Solution Explorer and click Publish.
    - In the "Target" tab, select Azure, and click Next.
    - In the "Specific target" tab, select "Azure Function App (Windows)", and click Next.
    - In the "Function Instance" tab, click on "Create a new Azure Function…". In the new dialog box:
        - Name – this needs to be globally unique, using only alphanumeric characters or a hyphen.
        - Subscription – an existing or new Azure subscription.
        - Resource Group – an existing or new Azure Resource Group which will contain the function.
        - Plan Type – select "Consumption" (serverless)
            - You pay only when the functions runs, based on number of executions, execution time and memory need.
            - However, there may be cold starts (initial delays when running the function)
            - The "Premium" plan has no execution code, but bills per second your functions are able to be called, together with the memory used. It reduces cold starts.
            - The "Dedicated" plan is billed as per other App Service Plans. It eliminates cold starts.
        - Location – where the function is going to be located.
        - Azure Storage – This new or existing account is necessary for a Functions runtime. Numbers and lowercase letters only, and the name needs to be 3 to 24 characters long.
        - Then click "Create".
    - Ensure that "Run from package file (recommend)" is selected.
        - This will select the Zip Deploy with run-From-package. This is recommended for Azure Functions deployments.
        - If you don't select this, it will use a Web Deploy – this is to any Internet Information Services (IIS) server,.
    - Click Finish, and then Publish.
    - The function's root URL will then be displayed on the Publish tab.
        - You can also click "Open in Azure portal", or "Manage Azure App Service settings".
- For editing (other than in the Azure portal):
    - You may be able to use the Azure portal.
    - You can also use Visual Studio for C# class library and script.
    - You can add the Azure Functions extension to VS Code, or for Linux, MacOS and Windows, and use C# class library and script, JavaScript, PowerShell and Python.
        - To install it, go to Extensions and search for azure functions, and then click Install. There will then be an Azure icon on the Activity bar, and an Azure Functions area in the Side bar.
        - You can then click Create Function icon, select a function and language, and click the HTTP trigger.

        - You can do the same for Azure Functions Core Tools for a Command prompt or terminal.
- To create a function in the Azure portal:
    - Go to the Azure Portal – and create a Function App.
    - On the Basics page, select:
        - Subscription,
        - Resource Group,
        - Function App Name (a globally unique name),
        - Publish – publish code files,
        - Runtime stack – either C# script, JavaScript, PowerShell or TypeScript.
        - Version number, and
        - Region.
    - On the Hosting page, select:
        - A Storage account,
        - Operating system – in-portal editing is only supported in Windows,
        - Plan – choose Consumption (Serverless),
    - We can skip the Monitoring and Tab pages.
    - Go to Review + create page, then click Create.
    - When it is deployed, click "Go to resource".
    - Go to Functions (on the left-hand side), and click "+ Add".
    - Select the appropriate trigger (e.g. HTTP trigger).
        - Name the function and the authorization level.
    - Click the function.
        - Edit the code as required.
        - Click on "Get Function Url", and copy it to the clipboard, and paste it to Notepad.
    - Go to the "Code + Test" page and expand the Logs.
    - Paste the URL into your browser's address bar.
        - Add &name=Phillip to the end.
        - Go to this web page.
- To remove the function:
    - Go to the function in the Azure portal.
    - In the Overview tab, click on the Resource group.
    - Click on "Delete resource group", enter the Resource group name, and click "Delete".
- To create the function in Visual Studio Code, you need:
    - .NET 6.0 SDK
    - Azure Functions Core Tools version 4.x.
    - C# extension for Visual Studio Code.
    - Azure Functions extension for Visual Studio Code.
- You then need to do the following:
    - Click on the Azure icon (on the left-hand side),
    - Then in the "Workspace Local", click the +, then "Create Function" – "Create new project".
    - Select where you are going to save the workspace.
    - Enter:
        - Language: C#,
        - .NET runtime: either .NET 6 or .NET 6 Isolated,
        - Template: HTTP trigger,

        - Function name and namespace,
        - Authorization level,
        - How to open your project: Add to workspace.
- To run it:
    - Click F5 to start the project.
    - In Workspace Local – Local Project – Functions, and right-hand click on the Function Name. Then click "Execute Function Now…"
    - In the Enter request body, press Enter.
    - Press Ctrl+C to stop the Core Tools.
- To publish it:
    - Click on Resources – Sign in to Azure, enter your credentials, and close the browser.
    - Click "+" next to Resources to create a resource. Enter:
        - Subscription,
        - Globally unique function name,
        - Runtime stack (C#) and
        - Location.
    - Then click Deploy (a cloud with an up-arrow).
    - Click "View output".
- To run it, go to Resources – Function App – [App Name] – Functions, right-hand click on the Function name and select "Execute Function Now…".

79. Authenticate to Microsoft Power Platform by using managed identities

- See topic 70.

Publish and consume Dataverse events

80. Publish an event by using the API

- See topics 48-50.

81. Publish an event by using the Plug-in Registration Tool

- See topics 65 and 67.

82/83. Register service endpoints: Azure Service Bus

- The Azure Service Bus:
    - This can relay message data between the Dataverse and Azure. Other applications can then get the messages from the Service Bus.
        - Allows for applications to be decoupled, improving reliability and scalability of applications and services.
    - Can be invoked from plug-ins for custom workflow activities.
    - Use it for high scale processing using (only) asynchronous steps.
- In Azure:
        - Create a Service Bus Namespace.
            - Connection String – go to the "Service Bus Namespace" – Settings – Shared access policies. Click on "RootManageSharedAccessKey" and copy the "Primary Connection String" (it will take a minute to be displayed).
        - Go to Entities – Queues, and create a new queue.
- In the Plug-in Registration Tool
    - Create the Connection with your Dataverse.
    - Register an Endpoint

        - Go to Register – Register New Service Endpoint.
        - Paste the Primary Connection String that you previously copied, and click Next.
        - Add the Queue Name, and the Message Format - .NETBinary, JSON or XML. Select JSON in this case.
        - Add UserId to the "User Information Sent".
    - Register a new Step
        - Go to your endpoint.
        - Go to Register – Register New Step
        - Enter the relevant data – for example:
            - Message: Create.
            - Primary Entity (Table): contact
            - Event Pipeline Stage of Execution: post-operation
            - Execution mode: Asynchronous
- Test the endpoint:
    - In the Dataverse, add a new Contact.
    - Go to the Service Bus Namespace, and check for the message OR Go to Settings – System – System Jobs, and see that it has been posted.

82/83. Register service endpoints: webhooks

- Webhooks
    - This is a lightweight HTTP way to connect Web APIs and other services with a publish/subscribe model.
        - The service will send the request and get the response.
        - It uses port 80 for HTTP and 443 for HTTPS.
        - It will have up to 60 seconds to get the response – otherwise, it will fail with 502 Bad Gateway, 503 Service Unavailable, or 504 Gateway Timeout
    - Can only scale up to the point that your web service can handle the messages.
    - Can be invoked from plug-ins or custom workflow activities.
    - Can have either synchronous or asynchronous steps.
- In the Plug-in Registration Tool
    - Create the Connection with your Dataverse.
    - Register an Endpoint
        - Go to Register – Register New Web Hook.
        - Enter:
            - The unique name,
            - Endpoint URL (you can use a service such as Request Bin – pipedream.com – or an Azure Function), and
            - Authentication – either
                - HTTPHeader – key values [Key1: Value, Key2: Value] in the Header of the HTTP request.
                - WebhookKey – uses ?code=value
                - This is useful with Azure Functions, as query strings are expected to have a key name of "code"
                - HttpQueryString – key values, such as ?KeyA=ValueA&KeyB=ValueB
    - Then register a new step:
        - Enter Message (e.g. Create) and Primary Entity (table)
            - For Asynchronous mode, go to Settings – System – System Job

- For synchronous mode, you need to link the next operation triggered by the WebHook to start immediately, or you get an Endpoint unavailable error.

82/83. Register service endpoints: Azure Event Hub

- Azure Event Hubs:
  - A highly scalable publish/subscribe service.
  - It can ingest millions of events per second.
  - It can stream the events into multiple applications.
- To create an Event Hub:
  - Create the Event Hub.
  - Then create an Event Hubs namespace:
    - Go to "+ Add" in the Event Hub.
    - Choose a Pricing Tier:
      - Basic gives you 1 consumer group per event hub – it costs around $11 per month. (Extra charges for more than 1 million events per month and for enhanced support).
      - Standard gives you 20 consumer groups, but only costs double per month (around $22 per month).
      - Premium and Dedicated give you 100 and 1,000 consumer groups. It costs $750 and $5,000 per month.
    - Click on "Create" the Namespace.
- Then create an event hub within the Namespace:
  - Click on "+Event Hub".
    - Enter a name.
    - The "partition count" allows parallelism.
    - The "message retention" show how long the service keeps data. It can be longer for the higher pricing tiers (1 day for Basic, 7 for Standard, and 90 for Premium and Dedicated).
- In the Plug-in Registration Tool
  - Create the Connection with your Dataverse.
  - Register an Endpoint
    - Go to Register – Register New Service Endpoint.
      - Use the Designation Type of "Event Hub".
      - Message body format: XML or JSON.
      - Use SAS authorization.
  - Add a step
  - Start your solution listening on the service endpoint.
  - Perform the relevant Dataverse operation.
  - Go to Settings – System – System Job to verify whether it has succeeded.

83. Implement a Dataverse listener for an Azure solution

- You can create a Dataverse listener in Azure using, among other apps, Azure Functions.
  - Use the appropriate trigger.
- You can create an Azure listener in Power Automate.
  - Create an automated flow.
  - Choose a flow's trigger by searching using the keyword "Azure".
    - There are many more triggers if you click on Skip and go to main authoring canvas.

- You can use:
  - Azure DevOps
    - When a work item is closed/updated/created
  - Azure Event Grid
    - When a resource event occurs
  - Azure Queues
    - When there are any/X number of messages in a queue
  - Azure Blob Storage
    - When a blob is added/modified
  - Azure IoT Central
    - When a rule is fired
- You can also use:
  - Service Bus (note: Not Azure Service Bus). To test this:
    - Create a Service Bus Namespace.
      - Connection String – go to the "Service Bus Namespace" – Settings – Shared access policies. Click on "RootManageSharedAccessKey" and copy the "Primary Connection String" (it will take a minute to be displayed).
      - Go to Entities – Queues, and create a new queue.
    - Go to Power Automate, and create a new automated flow.
      - Use the "Service Bus" – "When one or more messages arrive in a queue (auto-complete)" trigger.
      - Authentication Type: Access Key
      - Paste the Primary Connection key
      - Select the queue name
      - Add an action
      - Save and Test the flow
    - Go into Azure again:
      - Go into the queue, and go to "Service Bus Explorer (preview)"
      - Click on "Send messages" and send a message (Add something into the Message Body)
    - Go back into Power Automate
      - Go to the flow, and the latest run. Look at your action.

83. Implement a Dataverse listener for an Azure solution

- You can create a Dataverse listener in Power Automate.
  - Create an automated flow.
  - Choose a flow's trigger by searching using the keyword "Azure".
    - There are many more triggers if you click on Skip and go to main authoring canvas.
  - You can use:
    - Azure DevOps
      - When a work item is closed/updated/created
    - Azure Event Grid
      - When a resource event occurs
    - Azure Queues
      - When there are any/X number of messages in a queue
    - Azure Blob Storage

            - When a blob is added/modified
        - Azure IoT Central
            - When a rule is fired
    - You can also use:
        - Service Bus (note: Not Azure Service Bus). To test this:
            - Create a Service Bus Namespace.
                - Connection String – go to the "Service Bus Namespace" – Settings – Shared access policies. Click on "RootManageSharedAccessKey" and copy the "Primary Connection String" (it will take a minute to be displayed).
                - Go to Entities – Queues, and create a new queue.
            - Go to the Dataverse, and create a new table called MyMessages:
                - Add Content and Message id columns.
            - Go to Power Automate, and create a new automated flow.
                - Use the "Service Bus" – "When one or more messages arrive in a queue (auto-complete)" trigger.
                - Authentication Type: Access Key
                - Paste the Primary Connection key
                - Select the queue name
                - Add a Microsoft Dataverse – Add a new row action, and add "Content" and "Message id"
                - Save and Test the flow
            - Go into Azure again:
                - Go into the queue, and go to "Service Bus Explorer (preview)"
                - Click on "Send messages" and send a message (Add something into the Message Body)
            - Go back into Power Automate
                - Go to the flow, and the latest run. Look at your action.

84. Create an Azure Function that interacts with Microsoft Power Platform

- See topic 69.

Implement data synchronization

85. Configure table change tracking

- Table change tracking enables programmers to see what has changed since a point in time.
    - This means a program doesn't need to download the entire table each time it is opened, or needs refreshing.
- To enable a table, in the Power Apps portal:
    - Go to Dataverse – Tables – [Relevant table].
    - Go to Properties – Advanced options.
    - Select "Track changes", and click Save.
- Once change tracking has been enabled on a table, it is not possible to disable it.

86. Read table change records by using platform APIs

- To read changes to a table, you can use RetrieveEntityChangesRequest.
    - Sample code is in Github Code samples\cds\orgsvc\C#\Changetracking
- It includes the following properties:
    - EntityName – the table name,

- Columns – the columns to retrieve.
- DataVersion – uses a VersionToken.
  - If this is the first time you are using this function, do not include this. It will return all the records as new.
  - The VersionToken needs to be in the last 90 days – otherwise, all the records will be returned as new.
  - New/updated records will usually be returned before deleted records are returned.
- PageInfo – this includes:
  - Count – number of records to retrieve,
  - PageNumber – number of pages returned, and
  - ReturnTotalRecordCount – whether the total number of rows should be returned.

- You can then retrieve the VersionToken, and use it in the next RetrieveEntityChangesRequest to get anything which has changed.
- Example of syntax:

```
 string dataVersionToken;

List<Entity> initialRecords = new List<Entity>();

RetrieveEntityChangesRequest initialRequest = new RetrieveEntityChangesRequest()

{

EntityName = "sample_book",

Columns = new ColumnSet("sample_bookcode", "sample_name", "sample_author"),

PageInfo = new PagingInfo()

{ Count = 5000, PageNumber = 1, ReturnTotalRecordCount = false }

};

RetrieveEntityChangesResponse results =
(RetrieveEntityChangesResponse)service.Execute(secondRequest);

initialRecords.AddRange(initialResponse.EntityChanges.Changes.Select(x => (x as
NewOrUpdatedItem).NewOrUpdatedEntity).ToArray());

dataVersionToken = initialResponse.EntityChanges.DataToken;

 List<Entity> updatedRecords = new List<Entity>();

List<EntityReference> deletedRecords = new List<EntityReference>();

RetrieveEntityChangesRequest secondRequest = new RetrieveEntityChangesRequest()

{

EntityName = "sample_book",

Columns = new ColumnSet("sample_bookcode", "sample_name", "sample_author"),

PageInfo = new PagingInfo()

{ Count = 5000, PageNumber = 1, ReturnTotalRecordCount = false },

DataVersion = dataVersionToken

};

RetrieveEntityChangesResponse results =
(RetrieveEntityChangesResponse)service.Execute(secondRequest);
```

87. Create and use alternate keys

- Alternate keys could be used if:
  - You don't know the ID of the table row,
  - You have configured alternate keys for the table, and

- You have the relevant key values.

- You can use alternate key like this:

```
RetrieveRequest requestR = new RetrieveRequest()
            {
                ColumnSet = new ColumnSet("name", "fax"),
                Target = new EntityReference("account", "name", "My Account 11")
            };
            var responseR = (RetrieveResponse)service.Execute(requestR);
            Entity entityR = responseR.Entity;
            string faxNumber = "Nothing";
            if (entityR.Attributes.Contains("fax"))
            {
                faxNumber = (string)entityR["fax"];
            }
tracingService.Trace("Stage {0}", faxNumber);
```

- You can use a composite key (one which uses several columns) like this:

```
            var keyS = new KeyAttributeCollection();
            keyS.Add("name", "My Account 11");
            keyS.Add("anotherfield", "Another value");
            RetrieveRequest requestS = new RetrieveRequest()
            {
                ColumnSet = new ColumnSet("name", "fax"),
                Target = new EntityReference("account", keyS)
            };
            var responseS = (RetrieveResponse)service.Execute(requestS);
            Entity entityS = responseS.Entity;
            tracingService.Trace("Stage {0}", entityS["fax"]);
```