OrbitaAdirA Training-20250724_091156-Meeting Recording
July 24, 2025, 2:11PM
52m 30s

Mark Cline started transcription

Mark Cline 0:05
All right, all right, all right. OK. I'm going to turn it over to Andrew. Andrew, I I think we're going to be using which what? Adira Dev, is that the one?

Andrew Merola 0:17
Sure. Yeah. I guess I'm just wondering how much do we want to split between sort of just high level overview of the Orbita platform versus getting into the weeds on some of the sort of framework code?

Mark Cline 0:32
Let's do like I was gonna say, let's do 10 minutes or so on the overview and then we can get into the weeds. We have about 40 minutes left.

Andrew Merola 0:33
And.
OK, um, sounds good.
Let me share my screen here.
Everyone can see what I'm presenting.

Ameya Thete 0:58
Yes.

Andrew Merola 1:00
Yep, so this is the Orbitv platform. This interface here is what we refer to as Experience Manager. When you first log in, you're gonna see the listing of sort of all the projects that have been established with thinking that you know you.
You would have different projects for different customers, or perhaps sometimes different projects just for very distinct functionalities within the same customer. But yeah, so pretty basic. At the outset you'll have your list of projects clicking in to a project and I'll use the Adira Jade new.
As our example for this call, you're going to get project specific menus and so at a high level, Orbita was designed as a platform to enable the creation of skills once in our platform that can then.
Be deployed to a variety of sort of interactive um.
Skill frameworks, I guess you would say, with the main ones that we target being Google Dialogflow and Amazon Alexa Dialogflow being more our solution for sort of text chat bots. Alexa often being for, you know, the voice skills with other possibilities for like IVR and stuff, but that was the main, the main crux of the platform design it is.

Monica Chaudhari 2:32
Hey, sorry, Andrew. I just wanted to confirm that Mark has started the recording.

Andrew Merola 2:39
Uh, Yep, I I think he has.

Monica Chaudhari 2:39
OK. Excellent. Thank you.

Andrew Merola 2:42
Is that right, Mark?

Mark Cline 2:45
Yes, that's correct. I have started recording.

Andrew Merola 2:47
OK, yeah, so you know at a basic level it is, you know, a underlying it is a node JS application. You won't necessarily interact much with the node process if at all, but you know the the development is primarily JavaScript.
Based. Um, we hosted on AWS, but again, that's sort of, you know, not not something that you deal with at your level. Um.
And.
The primary backing of each orbiter sort of instance is it's built on top of a framework called Node RED. So this what I'm showing you here is Experience Designer. This is where you would write most of your.
Code for your projects. It's a visual editor. I'm not sure how else to describe it. It was primarily geared towards sort of Internet of Things applications.
And sort of a carryover from the earlier incarnation of Orbitome when I thought it was going to be a sort of healthcare IoT focused platform. It's certainly, you know, interesting being a visual editor allows you a lot of.
Sort of interesting possibilities for visualizing your flow logic, but at its core it's, you know, a series of nodes that you develop. A lot of these orange nodes, for example, are just sort of JavaScript function nodes.
It certainly, you know, encourages a lot of modularization of your code into very discrete units, which is good. And essentially when a flow in here by a flow I mean sort of some sort of starting node and then a series of nodes after it.
Is invoked. You just sort of get execution where what they call a message object is created and it gets passed through node to node until it hits an endpoint. But we'll get back to this in more detail later. So the primary loop in the orbit.
Platform is essentially a request comes in to one of these interactive frameworks like Alexa or Dialogflow, which will be our use case here. It comes in, it'll be saying an utterance of some kind. Google Dialogflow is what does. They do the heavy lifting in terms of sort of.

Interpreting the utterance into an intent based on what we have told it. So something like I want to schedule an appointment. Dialogflow interprets that as the scheduling intent. Once it has done that, it sends off a request to a webhook on the orbiter end saying hey.
The scheduling intent has fired and then Orbita will invoke an intent node like this in Node RED that will then carry on that message through whatever the configured code is for the scheduling intent.
Until it hits an endpoint and sends a response back that then gets sent back to Dialogflow and then back to the user. And you know, similar chain for Amazon Alexa would be a request comes into it, that request gets interpreted by Alexa sent over.
To the orbit of corresponding intent request node and it goes through our logic here in experience designer and then a response is sent back to the user. So the pieces that make that happen are at its basic level. Everything is sort of driven by intents.
Um. Intents are defined here in the intent library. Um. If we look into one, see if there's a good example here. Um.
This is pretty minimal at the moment, but you know if you wanted to create a a new intent. What you would be asked for is a variety of utterances. So you know, let's go back to the example of, say, like a scheduling intent. You'd be filling it potentially with, you know, utterances like, you know, I want to schedule an appointment, that sort of thing.
Or what we would call templated utterances where you know you have slot lists which are lists of particular known values.
So let's say um.
You know I want to schedule an appointment at, you know, downtown hospital or some location name. You would have a situation where you would have a list of say like scheduling location names and then you could create an intent here.
That would utilize that.
And just as an example here, um.
I'll just use the orbit of the built in sort of city slot. You know I can do something like you know I want to schedule an appointment in and then using the bracket notation you would select a slot and this sort of represents you know.
To Dialogflow, like catching any utterances like I want to schedule an appointment in any of the values that are in that slot. So yeah, between hard coded utterances and templated utterances, intents are where you are indicating to either Alexa or Dialogflow sort of what what sort of utterances.
As you're expecting from the user and how they should interpret them as an explicit intent of the user.
So when you have created your intents, there is a deployment process over here. This is where in this case in the Alexa, you are seeing that Orbita has sort of taken all of your intent information and converted it into the format that Alexa would expect for a skill deployment.
We don't certainly display the same thing over here, but Dialogflow, Google, same thing is happening there. We are converting it into the sort of format that Dialogflow would expect and then you are publishing out there for the project. So that that that agent that is listening for user input is then ready to interpret whatever sort of.

Utterances you have configured. And like I said, you know that's sort of the that builds the core loop of Orbita projects, user request, intent interpretation, sending that off to Orbita for processing, we give a response back to.
The agent that goes back to the user. So then there are a bunch of other mechanisms that sort of you know around building that sort of intent fulfillment logic. The I'll skip over some of these because they're not really that relevant to most project development.
But the big one would be Orbita Answers, which you know handles sort of your Q&A type situations. We do have some increasing stuff that we are building around generative AI handling question answering.
That's something we can get into more detail later, if applicable, and Flow Studio would be the other very big piece of intent for development. So what you see in Experience Designer is great for a lot of sort of like custom logic for, you know, let's say you need to.
Call out to some third party web service or retrieve records from within Orbita. We have a content library for storing any sort of data that you might want to store based on schemas that you define.
But it has some limitations in terms of flow logic, so that's where Flow Studio comes in. So in Flow Studio you will see the development of sort of user experience flows.
Going in here, this is a rather large one, but like this where you know and this was sort of developed as a more less technical user-friendly means of developing experiences. But in here is where you know you can sort of piece together nodes that.
That either presents, you know, content to the user or ask them for input. So this node here represents sort of, you know, asking for input and then presenting them choices for that input. There are nodes in here that.
We'll ask for sort of free form input. Here we are presenting a text box to the user asking for a zip code nodes that sort of connect back into Experience Designer for more complex evaluations.
For example, um.
Anyway, so this Flow Studio would be your sort of primary development of the actual user facing experience by and large with some back and forth between Flow Studio and Experience Designer depending on the complexity of what you were trying to do.
Um.
I guess one other thing to touch on quickly here would be adaptive cards. I don't know if there are any in this project, but it provides sort of an interface for creating more complex user interface or user like input interfaces.
So you know in Flow Studio you have sort of these simple options for you know say I want to just present a text box that takes in a zip code from the user. Adaptive cards provides tools for let's say you know creating like a multi input form of we want like first name, last name, e-mail, date of birth with a calendar selector.
From the user, so that's another useful tool, yeah.

Daniel Kravets 13:12
Just want to check from the installation side if there are any any questions about anything so far.

Andrew Merola 13:14

So.

Daniel Kravets 13:26
I guess not.

Andrew Merola 13:27
OK, so two other things that I'll just touch on quickly here. I had mentioned content. So you know there are a lot of mechanisms in Orbita for you know, like I said, storing custom data based on schemas that you define.
You know, taking a look at schemas here, for example, this is a schema for an instance of an error that has occurred that we want to store to sort of be able to review later. It's primarily just an array of field definitions where you say sort of what the key is.
Is what the type of field is, any validation requirements, and then that enables the storing of data either in what we call the content library here.
Yeah, let's see. I have a good example here. So in this case, one of our content sort of schemas is settings, where we store setting information for our project looking at the.
Error schema. It's over here under section called data. I'll get into why there are two different sections in a moment, but here we can see items that were records that were created based on that schema that I showed you over there of the.
Of the error records and you know we present sort of a, you know, dynamically generated interface for editing based on what the schema is. The items are split between 2 sections here. There was a distinction that was more important earlier on.
In Orbita. So initially there was content and there was data with. Mark correct me if I'm wrong, but I feel like content initially was the only thing that got indexed for search and data did not. Or it was maybe the other way around.
That distinction has sort of since been removed, so either data under the, you know, sort of content section or data section can be indexed for search. So really there isn't a significant difference between.
Which area you use for content storage? It will be determined by the type you select when creating the schema. So there is a content type which will go in this section, a dynamic type which will go in this section.
I think in general we sort of just enforce the convention of, OK, things that are under the content or things that we would expect might be edited by, you know, less technical users, let's say if it's content items that represent, you know, content that we display to the user, that sort of thing. Whereas data may be more for analytical sort of data errors that we're recording, user interactions that we're recording, that sort of thing. But at the end of the day, there's really no huge functional difference between content stored in one place over the other.
And your primary means of interacting with content from experience designer would be the dynamic data manager nodes where you sort of tell it which model you're trying to interact with. So let's say error for example.
And what you were trying to do with it, whether or not you were trying to create a new record, update an existing one, read it. You know, it's primarily analogous to sort of just CRUD operations against a database and the payload in that.

In that node would be essentially a Mongo query. Mongo DB is the backing database for for Orbida for most things.
And then one more thing that's important to touch upon here is chatbot settings. So not applicable in Amazon Alexa skills, but very entirely applicable in Dialogflow skills that are fronted by a chatbot. So if we pull up.
An example here.
Um, actually let me pull up.
OK, there are style sheets on here that are interfering with the default orbit chatbot display. And so essentially in the chatbot settings, by default when a project is created, you're gonna get this default chatbot that has default orbiter stylings for the chatbot that is displayed.
On the page, which you know is a good starting point, but you will often you know, just inevitably do a very customer specific branding which is represented here by these change chatbot settings which have sort of changed the default.
Orbita chatbot styling into this Jade specific styling. You can see here it sort of takes over half the screen. There's complete flexibility whether or not you want sort of a chatbot that just takes up a corner of the screen versus a full page view.
But the chatbot settings are what govern the display. You have the template itself for the chatbot, so you have a lot of control in what markup is actually rendered on the page, custom CSS that is applied to it.
Custom JavaScript that is invoked as part of the chatbot floating and then a variety of other settings in here depending on what what orbital functionalities may be utilized. This for example is where like live agent settings would live if that was being utilized for a project.
Yeah, so those I think are the key pieces at a high level in experience manager for your standard sort of Dialogflow based chat bot based Orbita projects.
You know, there are other items in here. They're just less commonly used. Things like, you know, calendars, surveys, provides another sort of alternative means for developing a customer like a user input survey, but it's sort of.
Fall into the wayside in favor of doing that through Flow Studio. Um.
Anyway, so I think that covers everything I want to cover in Experience Manager. So I guess I'll pause here again for any any questions about any of this.
Sure.
I

Akshay Pohandulkar 20:23
Down down some the board which we have developed the work and then we we want to down it. So hope we can do that.

Andrew Merola 20:32
I'm sorry, can you repeat that? I'm not sure I got the second part of the question.

Akshay Pohandulkar 20:35
Yeah. OK. Yeah. So, so we have, yeah. OK. So we have developed a developed one bot and deployed it and then we want to down it. So we want to unpublish it. So what will be the way to unpublish that bot?

Andrew Merola 20:42
Mm-hmm.
Uh, when you say unpublish it, you mean like sort of revert?

Akshay Pohandulkar 20:55
Yeah, yeah, right. We don't want it to open it for the public, so we just wanted to revert it. So is there any easy way to do it so that the code will be as it is? It will not get removed from the projects, but the?

Andrew Merola 20:55
To um.

Akshay Pohandulkar 21:10
But will get unpublished.

Andrew Merola 21:13
So I mean the actual live availability of the bot is gonna be dependent on whatever page it's being hosted on. So that would sort of be up. So you know, we don't.
So you know, we serve.

Akshay Pohandulkar 21:30
OK, so we need to, yeah, we have to remove the JavaScript which have we have placed on the hosted server and then it will get down automatically, right? Is there no any way there is no way to unpublish from the platform, right?

Andrew Merola 21:38
Yeah.
Yep.
I mean, it's an interesting question. You know, typically, yes, it would be. You would remove the scripts that is injecting the chat bot onto the hosted page. There certainly would be options if you wanted to enable toggling that from Orbiter to sort of create a flag on the Orbiter side that then.
And like you know every request comes in for the chat bot, you just sort of cut it off at that point. But yeah, primarily that sort of thing we would handle from the hosted page. But you you know there is flexibility to also just like I said have a you know live flag or something in the settings in orbiter that you could then you know look at and experience design.
and sort of cut off chat bot loading. Yeah.

Akshay Pohandulkar 22:30
Yeah, OK, got it. Thank you.

Andrew Merola 22:35

Yep. So you know, just to go into, you know, I guess it's a good segue into sort of the experience designer part of things, but like chat bot for example. So you know this request is coming in for the sort of an orbit out hosted.
Chatbot page. So that request is coming into this HTTP endpoint here that is displaying an HTML template that we have created where among other things, we are injecting the JavaScript to load the chatbot.
In an actual production situation, you know this page would be a, you know, actual hosted page on whatever the customer website is with, you know, the request to be made into the orbit script and everything.
Um.
But yeah, so HTTP endpoints one of the sort of many options for input into experience designer code. The primary ones are going to be intent requests like I mentioned, which these are handling input from Dialogflow based on received.
User utterances and the interpreted intent. HTTP endpoints are gonna be another very common one. You add one of these into the project, you tell it what the path is, and then it's exposing that as an endpoint for anyone to request from the project instance through this sort of the.
And then there's the OEAPI prefix for any of those sort of endpoints. You'll see endpoints like this, which is a Websocket input that handles.
Input back and forth from the actual chat bot interface. Um.
There are inject nodes which can be scheduled at a specific time to fire off a process in Experience Designer. They can, you know, be manual only if there's say some sort of like, I don't know, clean up routine or something that you have created in here to.
Remove some stale data or something that you only want to fire manually. Um.
I think those are the primary means of input that you would typically see in Experience Designer. But like I said, you know the sort of core unit of functionality in Experience Designer is some sort of flow that goes from some sort of input node through a series of logic.
To some sort of response node, whether or not it's an HTTP response node that is indicating like what the actual response body content type is and a web socket response, or in the case of intent nodes, they are typically going to flow into.
A response node that will create the appropriate format of response for Alexa or Dialogflow, depending on what's being used. A few other less commonly used ones, especially I think from what I've seen so far in the Adira projects, but.
You know there are Twilio nodes for inputs to and back to Twilio as an SMS handler.
Um.
Yeah, those would be the primary ones. So like I said, all you know, everything you're going to see in here is JavaScript based.
The.
I guess a note worth making is that so any sort of, I would say modern JavaScript code is going to work fine in Orbita. We at one point forked Node RED.
And lost some more recent updates in terms of what the JavaScript editor will recognize as valid JavaScript. So if you were to use, say, like async and await as keywords in your JavaScript code in here, it may throw an error about it.

It often you know also likes to throw errors about like the sort of expansion operator for expanding objects into their constituent properties and stuff, but it will all it will all function. The you know sort of backing node JS version has been upgraded over time, but they're.
The editor is due for an upgrade. Um.
But yeah, so it is. It is pure JavaScript. Um.
And.
Yeah, I guess there's not much else to say about that. Like I said, it, you know, sort of lends itself nicely to being very, you know, discreet in your functionality. I would typically recommend that, you know, as you are developing in here that you keep each function node as succinct as possible and name it well to indicate exactly what is happening.
Happening in that node, it is very easy to just, you know, sometimes fall into a habit of creating a function node, you know, chaining it to the next one, chaining it to the next one, not naming anything. And then when it comes time to debug something, you're just kind of going through node through node and taking a guess as to what lives where.
And you know, there are often jokes around here about how how clean I keep my code in Experience Designer and people can, you know, tell just from the look of it whether or not it is code that I wrote.
Um, but yeah, it's the visual editor is interesting and it provides, you know, opportunity for, I think, very clean organization of of your code. So take advantage of that.
So.
I guess going into sort of the common framework that we have put together. So Orbita as a platform, especially with this experience designer piece, it's very, you know, it's very much just a a sandbox, it's.
A very thin layer on top of sort of open, you know, just development in node JS. So there aren't really a lot of limitations as to what you can do and that you know earlier on led to a lot of. Variations in code that we would see between projects. So at a certain point we came up with what we sort of called our module system within Experience Designer to help make certain things repeatable and standardize the code that we were putting out in our own projects.
And so everything that you will see in this project, for example, after this customer success modules tab is part of that framework and we would typically, you know, recommend and you'll see a note on every one of these tabs.
Not to edit it. These are standard modules and we have a whole system for deploying out different versions of them. And obviously if sort of custom changes have been made into one of these modules, that breaks the ability to do that.
If you were to sort of customize, let's say the chatbot module here for some reason. If we on our end, like you know, saw some sort of issue that we're trying to resolve with our standard implementation or want to enhance the functionality here and then deploy that out, we would overwrite the changes in that case.
So if at all possible, changes shouldn't be made for these modules. They should be made elsewhere in the project on sort of custom tabs, and each module should allow a sufficient level of interactivity with its methods from other tabs.
And I'll get into that in a moment, but so a quick rundown on the sort of common modules you'll see one. This CS modules tab is just essentially it's the sort of equivalent of a package dot JSON file in node JS. It's indicating what modules from our system have been deployed.

What versions? Sort of what the primary dependencies are, and then your full deployed dependency tree and what the versions are. Going through from the basic level, everything is built on sort of this project context system.
This solves a problem that we commonly had where we would use a lot of timed injects nodes to sort of initialize code on project startup and it just became a headache to sort of get them to fire in the correct order.
If you're using time to inject nodes, you can sort of stagger them saying OK, this one fires like 1/10 of a second after the project loads and then another one fires 1/2 a second to try to get the order right. But this became a more formalized version of essentially waiting for dependencies between projects to resolve.
Not something you need to worry about the details about, but just an explanation as for what that is doing there and why you see it referenced in all of these other modules where they are getting the project context. This is just like on the module startup. It is using this other sort of core module to handle dependency resolution.
The most common ones you will see all the time are utilities. This is common utility functions that are utilized across different modules. Settings handles sort of translating our settings content here and the values in there and making it accessible within Orbiter.
Over in this context section and updating it here if changes happen to that content. And then the other sort of really core one would be error handling which handles on any of these other nodes if an error, an unexpected error were to happen.
Now we get funneled into there so we can create a record of it, and so there's some features in here around sending out notification emails when errors happen.
So you have those core sort of modules and then built on top of that you have you get into the more, you know, actual functionalities that you'll see in most projects, things like Flow Manager, which handles input to and advancing users through those Flow Studio flows.
And then the big one would be Chatbot, which handles the input into and responses back from Dialogflow to Chatbot users, as well as presenting sort of these Orbita hosted pages primarily used for.
Um, you know internal testing before you are ready to deploy out to a customer site.
Um.
There is also sort of, I guess another category we'd say of modules that we refer to usually as utility pages that presents a sort of framework of.
One second here. Additional sort of augmentations of stuff that you know are administrative tasks, but then aren't necessarily covered under Experience Manager. Open it up in a new tab here.
Um.
But yeah, so this essentially covers pages that we sort of saw a need for in customer success, but needed to get built out more quickly than they could be created as actual project features.
So the the big one is going to be version control. So by default in Orbita there isn't necessarily any built in version controls. So one of our sort of big augmentations that we did were was creating a system for that.
So there's a whole system in place that can be utilized that will take a look at the sort of current project code, all of it deconstructed into a format that would make sense if looking at it, looking at it.

It all in a git repository, checking that in, sort of moving it up between stage and production environments, reverting if needed. But this this would be a subject that we'd probably want to get into in another call to fully cover it.
But yeah, that's another sort of common, common thing that you will see among the modules, things under the sort of category of utility pages, examples being version control. Sometimes we'll push out one called business hours. That's just sort of a complex interface for editing. Business availability hours, one for content synchronization, but I guess you could say so those under the category of custom tools that are not inexperience manager, but that we saw the need for.
Yeah, so we have the module system. Like I said, the, you know, thinking behind it was standardize our code, be able to deploy it out cleanly like you would say packages within like a node project and all developed from the thought that OK, the, you know, people utilizing these modules shouldn't.
Need to add the modules themselves. They should be able to interact it with it from other nodes. So getting into that for a moment, taking Flow Manager as an example. So a lot of Flow Studio flows are gonna operate on.
Not sure if there's an example here, but operate on interactions with Experience Designer through custom nodes. This essentially passes control off to Experience Designer to do some more complex logic and pass it back to Flow Studio.
Um.
Initially there wasn't really a great mechanism for even maintaining these connections. So typically in experience designer you would sort of need to look for the control ID and handle it based on that. One thing we standardized was essentially using the custom node name as a function name.
And implemented all the code for handling that. But obviously the actual sort of hooks that we call them themselves are going to vary a lot project to project. So one example of interaction between the sort of code and different tabs is when the flow manager.
Module here gets initialized. It is establishing a series of available methods in this project context that we referred to earlier. And then over on a tab here this is where we have our custom code for all of our custom hooks.
That are getting that project context, getting a reference to the flow manager module from it and calling that method to register the hug. So in node red there are.
What we call link nodes um for moving logic between um.
Between one tab and another I could have.
Say a link out node here that some you know code flows into and then a link in over here and connect these two.
And that's perfectly fine for your custom code. But in terms of modules, if we had, say, links like that, so you'll see links within modules within a tab, but links like that between tabs wouldn't really work for code that might be replaced in a piecemeal manner.
So that's why you would want to use this pattern of essentially like exposing the functions in context for code in other tabs to call. It's going to be a much more resilient pattern. I guess I didn't touch on it, but one sort of other important construct in.
Node RED development is these contexts. Um, so.

This is for data shared either between nodes within the same tab or amongst all the tabs. So one context that they allow is the flow level context. So you will see code sometimes like flow dot set and a key.
And the object or other data. So that is establishing that is setting this project property in the flow level context. This code, any data here is only accessible by other nodes within the same tab and then there is also the.
Global context so here.
For example or I guess not in there, but so you will see code sometimes that is a similar global dot set or global dot get that is establishing or storing, retrieving, updating data in this global context. This is shared amongst.
All tabs in the project and worth noting because this is something to be cognizant of is technically there is one node red instance that backs all projects on an orbiter environment.
So this would technically be shared not only amongst all tabs within this project, but amongst all tabs on all projects, which could lead to the temptation to do some clever things of sharing certain code and data between multiple projects in the same environment, but I would.
I would strongly recommend against that, and I would say that you should keep things as segmented as possible between your projects, despite the fact that there are some of these gateways in the back end between different projects. Because you know it's it's not so it's going to be obvious, so you could.
Change something in this global context and have another project that's referencing it. You would not really know that, and you could unintentionally create issues there.
Part of the global context that gets populated by default by Orbita is, you know, a variety of libraries that may be useful to you in certain situations. There's crypto JSS library, Lodash is in there, moment and moment time zone.
Um.
There are some orbit specific utilities around things like I think there's a function here I can you can call to like get the name of the current project you're in, that sort of thing. So yeah, there's some potentially useful stuff in there for you.
Um.
And yeah, I think, I think that mostly covers the basics of the module system. Um.
I guess I would also just say it's worth taking some time to sort of, you know, familiarize yourself with the available nodes in Node RED. Some of them, you know, are very orbiter specific, some are just sort of built in interesting.
Logic nodes around, you know. Essentially it's a gives you a clean interface for sort of a switch statement of changing direction based on conditions. Every node you see in node red is gonna have.
One input, but a variable number of of outputs, which makes sense. You know more than one node can flow into any given node, but there's really just the one input.
And in the case of a function node, let's see if I can find a good example here. Well, so in the case of a function node, you know all code in here is going to be executed with the assumption that this this message object has been passed in from the prior node.
And is carrying some sort of payload to be carried through within a function node. I can customize it to say any number of outputs. The sort of standard return for a function node with one output would be just to return the message object.

If you don't have a line like this, your execution is just kind of die at that node. There's an alternative form of node dot send message which achieves the same thing, but this is more useful in sort of when you're writing asynchronous code.
And then in the case of multiple outputs it would be expecting an array. So if I do something like an array of message null, message null, this for example in the case with four inputs would be sending the message out on pins one and three here.
But not two and four. So you know the order here of what you're passing out corresponds to the order of your output pins.
Um.
There are some other operators here for, say, joins and splits. This split would operate on whatever the message payload is, and assuming it's an array would sort of split it out and invoke the next node once for each item that was in the array and then at the end.
If you do a join, it could sort of join those results all back into one payload for the next node.
There's some sorting options. There are some sort of conversion options from different formats, XML, Jason, that sort of thing.
Yeah, so I mean, pretty much anything you could want to do could certainly be handled by just, you know, only using JavaScript function nodes. But there are some, like I said, some other interesting options in here that can lend itself to some clean code.
So yeah, I think that's.
I think that's everything that it would. Well, that's everything we have time for right now anyway. So any any quick questions?

Ameya Thete 45:42
Andrew, I have one small question. Could you please tell our developers that which tabs or modules are important for them like to change where they will be doing most of the changes like custom hook and which are not like really they they should not focus it more like.

Andrew Merola 45:44
So.

Ameya Thete 46:00
They're more of a a one time activity.

Andrew Merola 46:04
Yeah. So typically like if you'll see all of the customer success modules typically at the end of the list of tabs here, the tabs can be reordered. So it's not necessarily a hard and fast rule, but every module.
Should be marked with a comment tag like this that says CS module do not edit. That would imply like you don't want to be writing code in here or touching this code really. So anything labeled custom. And again this isn't like a strict requirement, but you know if something is labeled custom, that's you know where you're going to be running custom project code.
But yeah, basically it's if it does not have one of these tags, then it's considered sort of custom code for that project from the module perspective.
Yeah.

So pay attention to this.

Ameya Thete 46:54
OK.
One last question, could you please drag the Dynamic Data Manager node?

Andrew Merola 47:01
Oh, sure.

Ameya Thete 47:04
If he.

Andrew Merola 47:04
And uh, one other sort of just like now it is.

Ameya Thete 47:06
Or you can clinic any if we have in a settings, we have that already, right?

Andrew Merola 47:12
Yeah, so in terms of getting a note here, like there is this list on the side. I could do like a filter up here for dynamic data. A very quick shortcut is if I control click anywhere on your grid here, it brings up this interface for quickly adding something in.
Yeah, so here's a dynamic data node.

Ameya Thete 47:30
Could you please click on like select the dynamic data node and click on information tab on right side in the content?

Andrew Merola 47:40
Oh, sure.

Ameya Thete 47:41
So earlier we had this help here.

Andrew Merola 47:46
Oh.

Ameya Thete 47:47
Which was used to come, which is not coming right now. Like all the like how read write everything was they used to show here.

Andrew Merola 47:50
Yeah.

Yeah, that is something I'll need to look into. Yeah, I'm seeing it for other nodes, so it's not across the board. But yeah, that's odd that that is no longer there. Yeah, I'll need to check with Aaron.

Ameya Thete 48:00
OK.
Hi.
Because in yeah in the morning for us like that orbital help was also not working that support manual that document we have and then that was also down I guess or something for us I don't know but this was I I seen like I I was.

Andrew Merola 48:13
Art.
Oh.

Ameya Thete 48:31
Giving some demo to my colleagues and I was not able to, you know, remember the syntax and everything. So it was a bit like if they see that there only then they can, you know, write it better. So yeah.

Andrew Merola 48:48
Yeah, no, that's that's definitely, yeah. Without that, you would have really no way of knowing sort of the intended use here. Yeah, I'll need to follow up on that. That's odd that that.

Ameya Thete 48:54
Bye, bye, bye, bye.

Andrew Merola 49:02
Would have gone away.
But yes, so typically what you would see like here in the function node, you'll see it where like there's a whole sort of in this info tab, there's a whole sort of information about actually utilizing it. You should see that for most nodes in here, and there used to be.

Ameya Thete 49:11
Right.

Andrew Merola 49:22
Health information here indicating sort of expected queries around different operations. So yeah, I'll I'll have our team figure out why that is no longer showing up there.

Ameya Thete 49:34
OK. That will be helpful, yeah.
Thanks.

Mark Cline 49:40
So couple takeaways, we'll check on the why the help's not working in the documentation. We'll send the recording to you as well on any other documentation types we've got kind of on the module process. I think it'd be good to schedule a follow-up call next week.
Does the same time work for everybody next week?

Ameya Thete 50:05
Is it possible like early next week?

Mark Cline 50:08
By Tuesday.

Ameya Thete 50:10
Yeah, that will be helpful.

Mark Cline 50:13
Andrew, does that work for you?

Andrew Merola 50:16
Um, let me just check quickly Tuesday at 10.

Mark Cline 50:20
Yes.

Andrew Merola 50:21
Yeah, that's fine.

Mark Cline 50:22
OK. So I'll send out a meeting invite for Tuesday at 10 Eastern. And if you have any questions beforehand, if you want to send those over, then we can be ready for those and can talk about those in more detail on Tuesday's call.

Ameya Thete 50:40
OK, but as Mark earlier I told you that.

Akshay Pohandulkar 50:43
At least I'm gonna.

Ameya Thete 50:43
Uh.
Sorry.

Akshay Pohandulkar 50:45
Yeah, I miss.

Ameya Thete 50:47
Yeah. So Mark like mentioned it multiple times that from our like we we always face issue of this modular, modular structure. Is it possible for us to you know cover like how flow manager works and how we can capture the hooks here?
And store data like in a model way.

Mark Cline 51:09
Yeah, yeah, we can talk about how to do your own.

Ameya Thete 51:11
Yeah. So if that, yeah, that will be helpful for team to, you know, go ahead with the new project which is coming in. So yeah, we will try to keep that as a target and like team will, you know. Meanwhile, they will go through all the training materials and this video, what they have learned today and they will, you know, come up with the questions which will they have. We will e-mail you that or yeah.

Mark Cline 51:44
OK.
Awesome. Well, very good. Thank you all for attending today. Thank you, Andrew, for doing the training. We'll send this out and we'll look at these a couple issues that you mentioned and we'll get ready for Tuesday's call.

Ameya Thete 51:49
Yeah, thanks.
Thanks.

Akshay Pohandulkar 52:06
Yeah. Thank you.

Daniel Kravets 52:07
Thank you, Mark and Andrew.

Ameya Thete 52:07
Thanks.

Akshay Pohandulkar 52:09
Yeah.

Mark Cline 52:10
Thanks everybody. I hope you all have a wonderful evening or day depending on where you are located and we will talk to you soon.

Daniel Kravets 52:17

All right. Same to you, Mark. Take care, guys. Bye.

Ameya Thete 52:17
Bye. Have a good day.

Andrew Merola 52:22
Bye, bye.

Mark Cline stopped transcription