
SynertyPeek Documentation

Release 3.1.0

Synerty

Apr 26, 2021

Contents:

1	How to Use Peek Documentation	1
2	Overview	3
2.1	Architecture	3
2.2	Services	4
2.2.1	Peek Logic Service	5
2.2.2	Storage Service	5
2.2.3	Field Service	5
2.2.4	Field App	6
2.2.5	Office Service	7
2.2.6	Office App	8
2.2.7	Worker Service	9
2.2.8	Agent Service	9
2.2.9	Admin App	9
2.3	Plugins	9
2.3.1	Enterprise Extensible	10
2.3.2	One Plugin, One Package	11
2.4	Noop Plugin Example	14
3	Tutorials	17
3.1	Peek Plugin Tutorial	17
3.1.1	First Steps	17
3.1.2	Scaffolding From Scratch	18
3.1.3	Add Documentation	26
3.1.4	Add Logic Service	31
3.1.5	Add Agent Service	35
3.1.6	Add Storage Service	38
3.1.7	Add Admin App	48
3.1.8	Add Field App	52
3.1.9	Add Field Service	56
3.1.10	Add Office App	61
3.1.11	Add Office Service	63
3.1.12	Add Tuples	66
3.1.13	Add Tuple Loader	73
3.1.14	Add Offline Storage	85
3.1.15	Add Observables	87
3.1.16	Add Actions	100

3.1.17	Add Vortex RPC	114
3.1.18	Add Plugin Python API	125
3.1.19	Add Plugin TypeScript APIs (TODO)	131
3.1.20	Use Plugin APIs (TODO)	131
3.1.21	Add Worker Service	131
3.1.22	Challenges	137
3.2	Twisted Python Tutorial	139
3.2.1	Understanding Twisted	139
3.2.2	Understanding Deferreds	144
3.2.3	Twisted Challenges	169
3.2.4	Understanding VortexPy	173
3.3	Python Gotchas	180
3.3.1	List Iteration Modification	180
3.3.2	Mutable Class Variables	181
3.3.3	Mutable Default Function Arguments	183
3.4	ReStructuredText Cheat Sheet	184
3.4.1	Sections	184
3.4.2	Header 2	184
3.4.3	Inline Markups	185
3.4.4	Files	185
3.4.5	Reference Links	185
3.4.6	URL Link	186
3.4.7	Code Block	186
3.4.8	Bullets	186
3.4.9	Numbered Lists	186
3.4.10	Images	187
3.4.11	Admonitions	187
3.4.12	TOC tree	187
3.4.13	Docstring Format	188
4	Setup OS Requirements	191
4.1	Setup OS Requirements Windows	191
4.1.1	Installation Objective	191
4.1.2	OS Commands	192
4.1.3	Installation Guide	192
4.1.4	Create Peek OS User	192
4.1.5	MS .NET Framework 3.5 SP1	193
4.1.6	Visual C++ Build Tools 2015	193
4.1.7	Setup Msys Git	193
4.1.8	Install PostgreSQL	194
4.1.9	Install Python 3.6	198
4.1.10	Install Worker Dependencies	200
4.1.11	Install Oracle Client (Optional)	201
4.1.12	Install FreeTDS (Optional)	202
4.1.13	What Next?	203
4.2	Setup OS Requirements macOS	203
4.2.1	Installation Objective	203
4.2.2	Installation Guide	204
4.2.3	Set Terminal shell to Bash	204
4.2.4	Safari Open Safe Files	204
4.2.5	Create Peek Platform OS User	205
4.2.6	Install Xcode	206
4.2.7	Install Homebrew	206
4.2.8	Preparing .bash_profile	207

4.2.9	Install Python 3.9.1	207
4.2.10	Install PostgreSQL	209
4.2.11	Install Worker Dependencies	213
4.2.12	Install Oracle Client (Optional)	215
4.2.13	Change Open File Limit on macOS	216
4.2.14	What Next?	216
4.3	Setup OS Requirements RHEL	216
4.3.1	Installation Objective	216
4.3.2	Installation Guide	217
4.3.3	Install Red Hat Linux Server 7.7 OS	217
4.3.4	Login as Peek	232
4.3.5	Registering RHEL	232
4.3.6	Removing IPv6 Localhost	232
4.3.7	Installing General Prerequisites	233
4.3.8	Installing VMWare Tools (Optional)	235
4.3.9	Update Firewall	235
4.3.10	Preparing .bashrc	236
4.3.11	Compile and Install Python 3.9.1	236
4.3.12	Install PostgreSQL	237
4.3.13	Install Worker Dependencies	242
4.3.14	Install Oracle Client (Optional)	243
4.3.15	Install FreeTDS (Optional)	244
4.3.16	What Next?	245
4.4	Setup OS Requirements Debian	245
4.4.1	Installation Objective	245
4.4.2	Installation Guide	246
4.4.3	Install Debian 10 OS	246
4.4.4	SSH Setup	251
4.4.5	Login as Peek	252
4.4.6	Configure Static IP (Optional)	252
4.4.7	Installing General Prerequisites	252
4.4.8	Installing VMWare Tools (Optional)	254
4.4.9	Preparing .bashrc	254
4.4.10	Compile and Install Python 3.9.1	255
4.4.11	Install PostgreSQL	257
4.4.12	Install Worker Dependencies	261
4.4.13	Install Oracle Client (Optional)	262
4.4.14	Install FreeTDS (Optional)	263
4.4.15	What Next?	264
4.5	What Next?	264
5	Deploy Peek Release	265
5.1	Windows	265
5.1.1	Deploy Virtual Environment	265
5.2	Linux	267
5.3	macOS	268
5.4	Development Considerations	268
5.5	What Next?	268
6	Administration	269
6.1	Configuring Platform config.json	269
6.1.1	Peek Logic Service	269
6.1.2	Peek Field Service	270
6.1.3	Peek Office Service	271

6.1.4	Peek Agent Service	271
6.1.5	Peek Field, Office, Logic Service SSL	272
6.2	Run Peek Manually	273
6.2.1	Check Environment	273
6.2.2	run_peek_logic_service	273
6.2.3	run_peek_office_service	273
6.2.4	run_peek_agent_service	274
6.2.5	Whats Next	274
6.3	Logs	274
6.4	Getting Support	274
6.4.1	Enterprise Support	275
6.4.2	Reporting Bugs	275
6.5	Updating Plugin Settings	276
6.6	Peek Windows Admin	276
6.6.1	Windows Services	276
6.6.2	Backup and Restore PostgreSQL DB	276
7	Capacitor App	279
7.1	Build Capacitor App	279
7.1.1	Install Capacitor	279
7.1.2	Build iOS App	279
7.1.3	Build Android App	280
7.1.4	Build Windows App	280
7.1.5	Further Reading	280
7.1.6	What Next?	280
7.2	What Next?	280
8	Peek Development	281
8.1	Develop Peek Plugin Guides	281
8.1.1	Develop Peek Plugins	281
8.1.2	Setup Plugin for Development	286
8.1.3	Developing With The Frontends	287
8.1.4	Continue Development	291
8.1.5	What Next?	292
8.2	Publish Peek Plugins	292
8.2.1	Create Private Plugin Release	292
8.2.2	Create PyPI Public Release	292
8.2.3	What Next?	294
8.3	Package Peek Plugins	294
8.3.1	Packaging a Production Release	294
8.3.2	What Next?	296
8.4	Develop Peek Platform	296
8.4.1	Development Setup Objective	297
8.4.2	Hardware Recommendation	297
8.4.3	Software Installation and Configuration	297
8.4.4	What Next?	301
8.5	Publish Peek Platform	301
8.5.1	Building a Production Release	301
8.5.2	Building a Development Release	302
8.5.3	What Next?	303
8.6	Package Peek Platform	303
8.6.1	Building a Windows Release	303
8.6.2	Building a Linux Release	304
8.6.3	Building a macOS Release	304

8.6.4	What Next?	304
8.7	Setup Pycharm IDE	304
8.8	Setup VS Code IDE	309
8.9	Helper Scripts for Development	309
8.9.1	git_check_sync.py	309
8.9.2	git_check_branch.sh	310
8.9.3	git_optimize.sh	310
8.10	What Next?	311
9	Troubleshooting	313
9.1	Troubleshooting Windows	313
9.1.1	Test cx_Oracle in Python	313
9.2	Troubleshooting Debian	313
9.2.1	Test cx_Oracle in Python	313
9.2.2	OSError: inotify instance limit reached	314
9.3	Troubleshooting macOS	314
9.3.1	Test cx_Oracle in Python	314
9.3.2	ORA-21561: OID generation failed	314
10	Utilities	317
11	API Reference	319
11.1	File plugin_package.json	319
11.2	SynerityPeek	319
11.2.1	peek_plugin_base	319
12	Release Notes	335
12.1	PEEP Overviews	335
12.1.1	PEEP 0-10	335
12.2	v3.1.x Release Notes	338
12.2.1	Platform Changes	338
12.2.2	Major Plugin Changes	338
12.2.3	Deployment Changes	338
12.2.4	Migration Steps	339
12.2.5	User Acceptance Test Results	339
12.2.6	v3.1.0 Resolved Issues Log	340
12.3	v3.0.x Release Notes	342
12.3.1	Platform Changes	342
12.3.2	Major Plugin Changes	342
12.3.3	Deployment Changes	342
12.3.4	Migration Steps	343
12.3.5	v3.0.0 Issues Log	343
12.4	v2.5.x Release Notes	345
12.4.1	Platform Changes	345
12.4.2	Major Plugin Changes	346
12.4.3	Deployment Changes	346
12.4.4	Migration Steps	347
12.4.5	v2.5.4 Issues Log	347
12.4.6	v2.5.3 Issues Log	347
12.4.7	v2.5.2 Issues Log	347
12.4.8	v2.5.1 Issues Log	348
12.4.9	v2.5.0 Issues Log	348
12.5	v2.4.x Release Notes	348
12.5.1	Platform Changes	348
12.5.2	Major Plugin Changes	349

12.5.3	Deployment Changes	349
12.5.4	Migration Steps	350
12.5.5	v2.4.5 Issues Log	351
12.5.6	v2.4.4 Issues Log	351
12.5.7	v2.4.3 Issues Log	351
12.5.8	v2.4.2 Issues Log	351
12.5.9	v2.4.1 Issues Log	352
12.5.10	v2.4.0 Issues Log	352
12.6	v2.3.x Release Notes	353
12.6.1	Platform Changes	353
12.6.2	Plugin Changes	353
12.6.3	Deployment Changes	353
12.6.4	Migration Steps	354
12.6.5	v2.3.3 Issues Log	354
12.6.6	v2.3.2 Issues Log	355
12.6.7	v2.3.1 Issues Log	355
12.6.8	v2.3.0 Issues Log	355
12.7	v2.2.x Release Notes	355
12.7.1	Platform Changes	355
12.7.2	Plugin Changes	355
12.7.3	Deployment Changes	356
12.7.4	Migration Steps	356
12.7.5	v2.2.2 Issues Log	359
12.7.6	v2.2.1 Issues Log	359
12.7.7	v2.2.0 Issues Log	359
12.8	v2.1.x Release Notes	361
12.8.1	Platform Changes	361
12.8.2	Plugin Changes	361
12.8.3	Deployment Changes	361
12.8.4	Migration Steps	362
12.8.5	v2.1.7 Issues Log	362
12.8.6	v2.1.6 Issues Log	363
12.8.7	v2.1.5 Issues Log	363
12.8.8	v2.1.4 Issues Log	363
12.8.9	v2.1.3 Issues Log	363
12.8.10	v2.1.2 Issues Log	363
12.8.11	v2.1.1 Issues Log	364
12.8.12	v2.1.0 Issues Log	364
12.9	v2.0.x Release Notes	365
12.9.1	Platform Changes	365
12.9.2	Plugin Changes	366
12.9.3	Deployment Changes	367
12.9.4	Migration Steps	367
12.9.5	v2.0.5 Issues Log	370
12.9.6	v2.0.4 Issues Log	371
12.9.7	v2.0.3 Issues Log	371
12.9.8	v2.0.1 Issues Log	372
12.9.9	v2.0.0 Issues Log	373
12.10	v1.3.x Release Notes	374
12.10.1	Changes	374
12.10.2	Linux Deployment	375
12.10.3	macOS Deployment	376
12.10.4	iOS Deployment	376
12.10.5	Windows Deployment	376

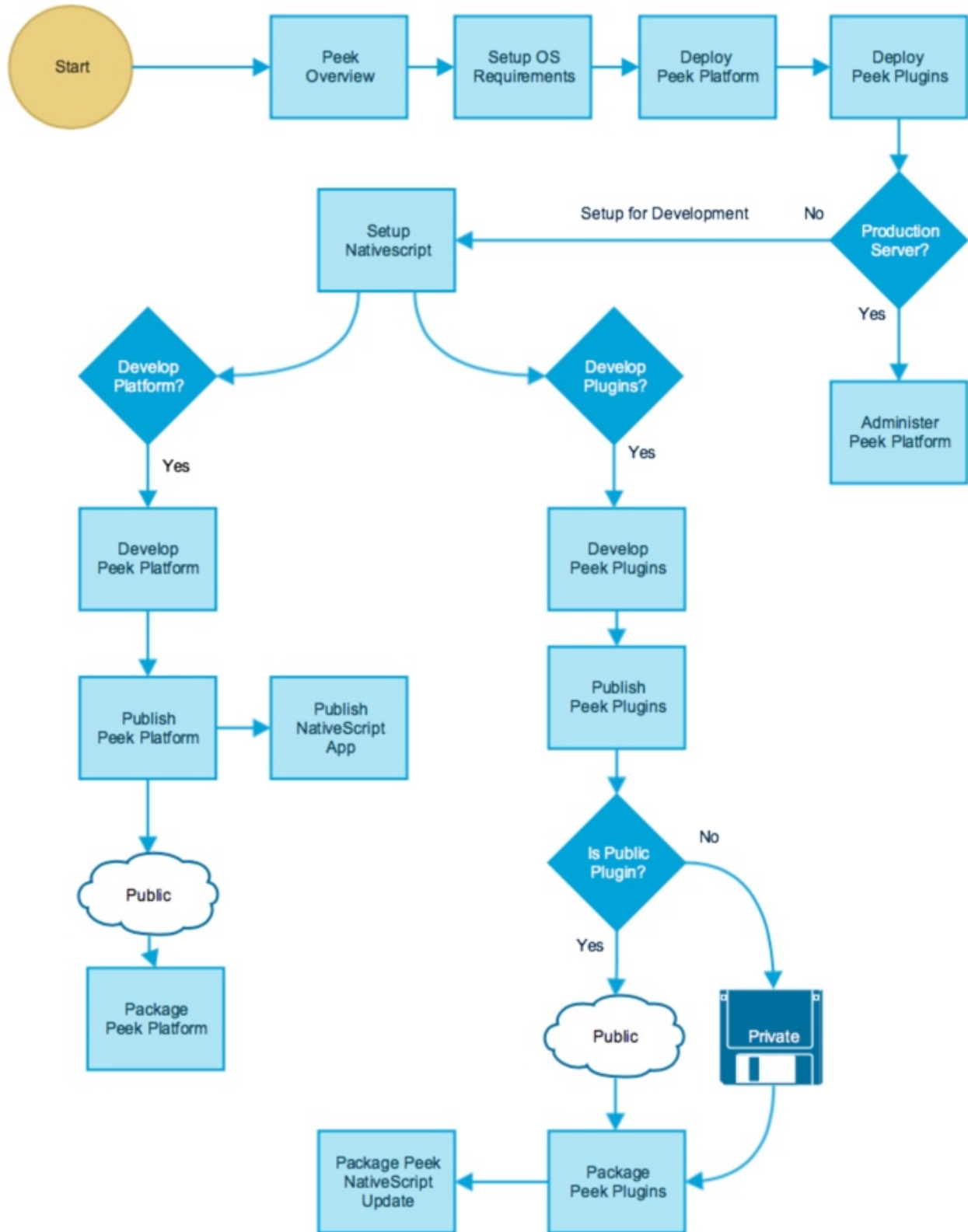
12.10.6	Enable New Plugins	376
12.11	v1.2.x Release Notes	376
12.11.1	Changes	376
12.11.2	Linux Deployment	377
12.11.3	macOS Deployment	377
12.11.4	iOS Deployment	378
12.11.5	Windows Deployment	378
12.11.6	Enable New Plugins	378
12.12	v1.1.0 Release Notes	378
12.12.1	Changes	378
12.12.2	Linux Deployment	380
12.12.3	macOS Deployment	380
12.12.4	Windows Deployment	380
12.12.5	Enable New Plugins	380
12.13	v0.10.0 Release Notes	381
12.13.1	Changes	381
12.13.2	Deployment	381
12.14	v0.6.0 Release Notes	382
12.14.1	NPM : peek-field-app-util	382
13	Indices and tables	385
	Python Module Index	387
	Index	389

CHAPTER 1

How to Use Peek Documentation

The Peek platform documentation is designed like code (IE, Modular).

Each blue square represents a document, follow this flow diagram to streamline your use of the Peek documentation.



CHAPTER 2

Overview

Peek Platforms primary goal is to manage, run and provide services to, hundreds of small units of code. We call these units of code plugins.

These plugins build upon each others functionality to provide a highly maintainable, testable, and enterprise grade environment.

Plugins can publish APIs for other plugins to use, and one plugin can run across all services in the platform if it chooses.

The Peek Platform provides low level services, such as data transport, database access, web server, etc. It effectively just bootstraps plugins.

With the Peek Platform up and running, plugins can be added and updated by dropping zip files onto the peek admin web page. The platform then propagates the new plugin, loads and runs it.

Higher level functionality is added by creating plugins.

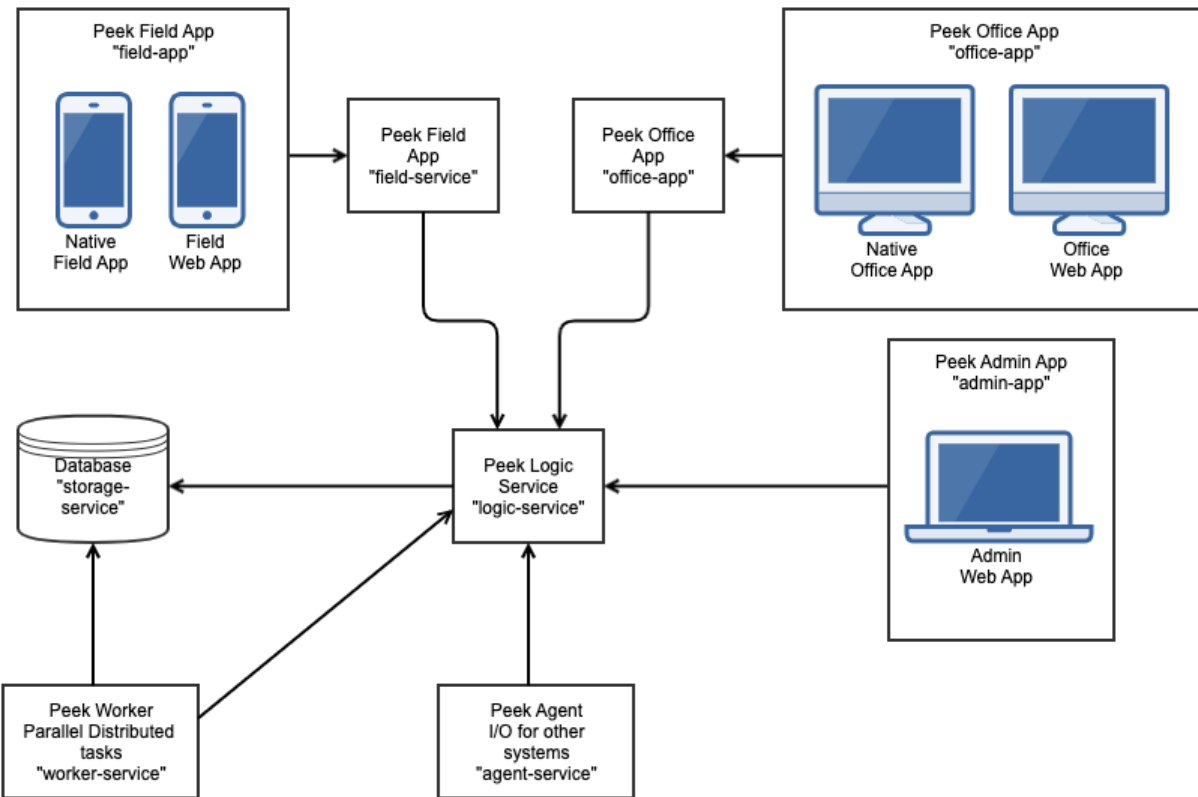
2.1 Architecture

The platform is distributed across several services, these services can be run all on one server, or distributed across different hardware and split across firewalls.

Peek supports distribution across multiple servers and network segregation.

For example, if you want to provide a means of integrating with external, less secure systems, you can place a “Peek Agent Service” in a DMZ to interface with the less secure networks. The Peek Agent will talk upstream to the Peek Logic Service.

The following diagram describes the architecture of the platform and the services it provides.



2.2 Services

This section describes the services which peek platform provides.

We use the term “service” with the meaning “the action of helping or doing work for someone”. Each service is it’s own entity which plugins can choose to run code on.

The exception is the “storage” service. The database can be accessed from the worker and logic services. The database upgrade scripts are run from the “logic” service. You could consider the database server to be the storage service.

Each service has it’s logical place with in the architecture. (See the architecture diagram above)

The services are as follows:

Table 1: Peek Platform Services

Service	Language	Description
logic service	python	The center of the Peek Platform, ideal for central logic.
storage service	python	This refers to support for persisting and retrieving database data.
field service	python	The field service handles field requests from ‘field app’.
office service	python	The office service handles office requests from ‘office app’.
agent service	python	The agent is a satellite service, integrating with external systems.
worker service	python	The worker service provides parallel processing for computational intensive tasks
admin app	typescript	A web based admin interface for the peek platform
field app	typescript	The user interface for web and native apps on mobile devices.
office app	typescript	The user interface for desktops

Note: Where we refer to “Angular” this means Angular version 2+. Angular1 is known as “AngularJS”

2.2.1 Peek Logic Service

The Peek Logic Service is the central / main / core service in the peek architecture. This is the ideal place for plugins to integrate with each other.

All other python services talk directly to this service, and only this service.

The main coordinating logic of the plugins should run on this service.

2.2.2 Storage Service

The storage service is provided by a SQLAlchemy database library, supporting anywhere from low level database API access to working with the database using a high level ORM.

Database schema versioning is handled by Alembic, allowing plugins to automatically update their database schemas, or patch data as required.

The database access is available on the Peek Worker and Peek Logic services.

2.2.3 Field Service

The Field service was introduced to handle all requests from native and web Field Apps. Reducing the load on the Logic Service.

Multiple Field services can connect to one Logic service, improving the maximum number of simultaneous users the platform can handle.

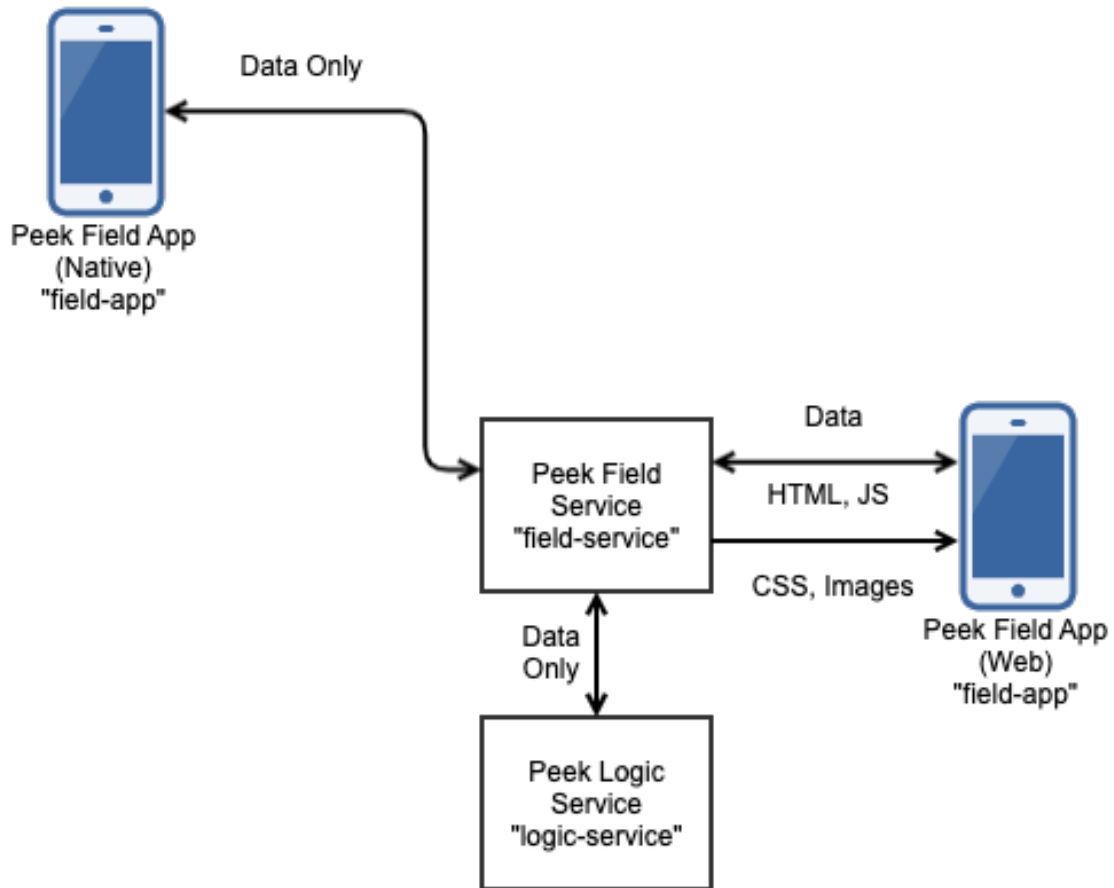
The Peek Field service handles all the live data, and serves all the resources to the Peek Field App.

The live data is serialised payloads, transferred over HTTP or Websockets. This is the VortexJS library at work.

The Field service buffers observable data from the Logic service. The Field service will ask the Logic service for data once, and then notifyDeviceInfo multiple users connected to the Field service when the data arrives. However, Plugins can implement their own logic for this if required.

The Field serves all HTTP resources to the native and web Field Apps, this includes HTML, CSS, Javascript, images and other assets.

The following diagram gives an overview of the Field Service communications.



2.2.4 Field App



The Field App provides two user interfaces, a native mobile app backed by Capacitor + Angular, and an Angular web app.

VortexJS provides data serialisation and transport to the Peek Field service via a websockets or HTTP connection.

VortexJS provides a method for sending actions to, and observing data from the Peek Field service. Actions and observer data can be cached in the web/native app, allowing it to work offline.

In web developers terminology, the Field App service is called the frontend, and the Field service is called the backend.

The Field service codes structure allows Angular components to be reused to drive web based interfaces. For example:

- **my-component.ts** (Angular component, written in Typescript)
- **my-component.web.html** (View for Browser HTML)

2.2.5 Office Service

The Office service was introduced to handle requests from native and web Office Apps. Reducing the load on the Logic Service.

Multiple Office services can connect to one Logic service, improving the maximum number of simultaneous users the platform can handle.

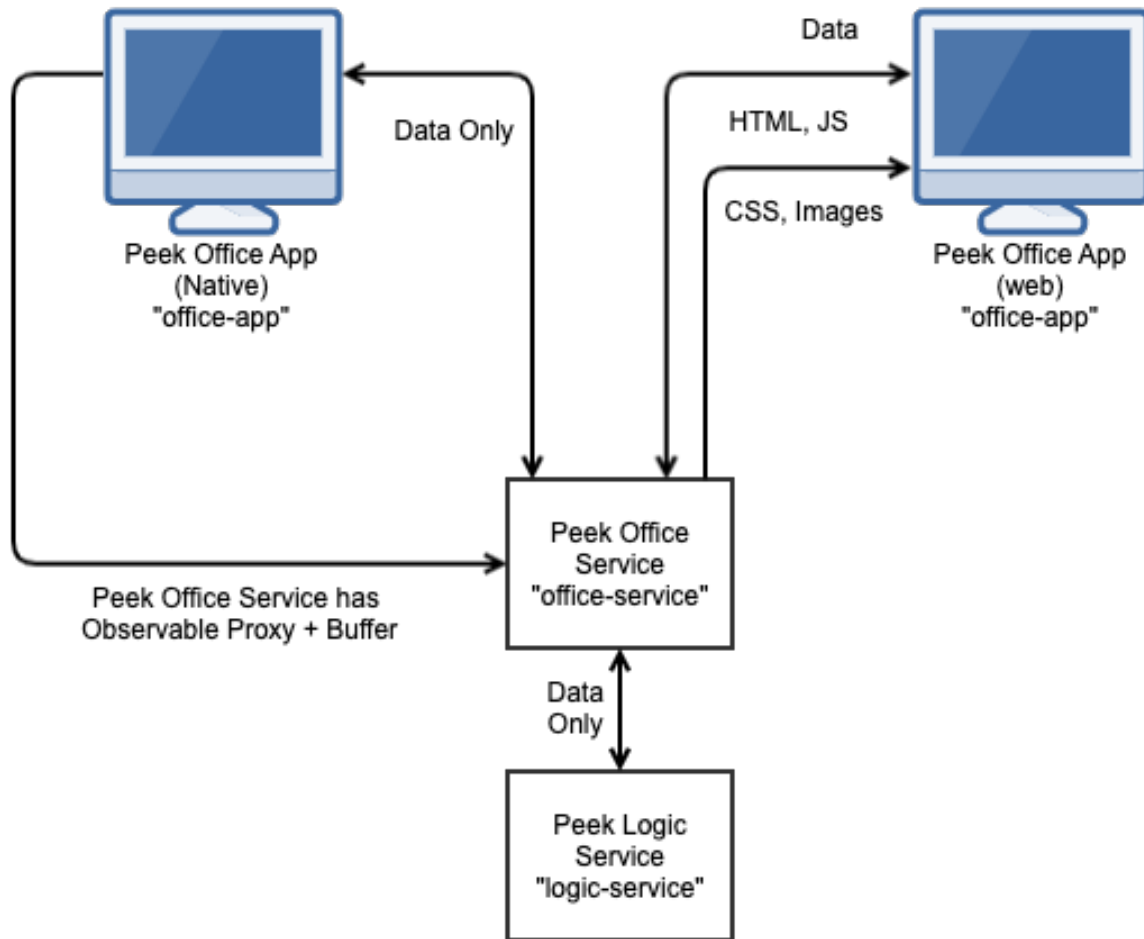
The Peek Office service handles all the live data, and serves all the resources to the Peek Office App.

The live data is serialised payloads, transferred over HTTP or Websockets. This is the VortexJS library at work.

The Office service buffers observable data from the Logic service. The Office service will ask the Logic service for data once, and then notifyDeviceInfo multiple users connected to the Office service when the data arrives. However, Plugins can implement their own logic for this if required.

The Field serves all HTTP resources to the native and web Office Apps, this includes HTML, CSS, Javascript, images and other assets.

The following diagram gives an overview of the Office Service communications.



2.2.6 Office App



The Peek Office app is almost identical to the Field app, using Electron + Angular for Native office apps and Angular for the web app.

The Office service has a different user interface, designed for desktop use.

The Office service code structure allows Angular components to be reused to drive both electron and web based interfaces. For example :

- **my-component.tron.html** (View for Nativescript XML)
- **my-component.ts** (Angular component, written in Typescript)
- **my-component.web.html** (View for Browser HTML)

Plugins can be structured to reuse code and Angular components between the Field and Office services if they choose.

2.2.7 Worker Service

The Peek Worker service provides parallel processing support for the platform using the Celery project.

The Worker service is ideal for computationally or IO expensive operations.

The Peek Logic Service queues tasks for the Worker service to process via a rabbitmq messaging queue, the tasks are performed and the results are returned to the Peek Service via redis.

Tasks are run in forks, meaning there is one task per an operating system process, which achieves better performance.

Multiple Peek Worker services can connect to one Peek Logic Service.

2.2.8 Agent Service

The Peek Agent service provides support for integrations with external system.

The Agent allows Peek to connect to other systems. There is nothing special about the agent implementation, it's primary purpose is to separate external system integrations from the Peek Logic service.

Peek Agent can be placed in other networks, allowing greater separation and security from Peek Logic.

Here are some example use cases :

- Querying and update Oracle databases.
- Providing and connecting to SOAP services
- Providing HTTP REST interfaces
- Interfacing with other systems via SSH.

2.2.9 Admin App

The Peek Admin app is the Peek Administrator user interface, providing administration for plugins and the platform.

The Peek Admin App is almost identical to the Field and Office Apps, however it only has the web app.

The Peek Admin service is an Angular web app.

2.3 Plugins

The Peek Platform doesn't do much by itself. It starts, makes all it's connections, initialises databases and then just waits.

The magic happens in the plugins, plugins provide useful functionality to Peek.

A plugin is a single, small project focuses on providing one feature.

2.3.1 Enterprise Extensible

The peek platform provides support for plugins to share the APIs with other plugins.

This means we can build functionality into the platform, by writing plugins. For example, here are two publicly release plugins for Peek that add functionality :

- Active Task Plugin - Allowing plugins to notifyDeviceInfo mobile device users
- User Plugin - Providing simple user directory and authentication.

The “Active Task plugin” requires the “User Plugin”.

Plugins can integrate with other plugins in the following services:

Table 2: Peek Plugin Integration Support

Service	Plugin APIs
logic service	YES
storage service	no
field service	YES
office service	YES
agent service	YES
worker service	no
admin app	YES
field app	YES
office app	YES

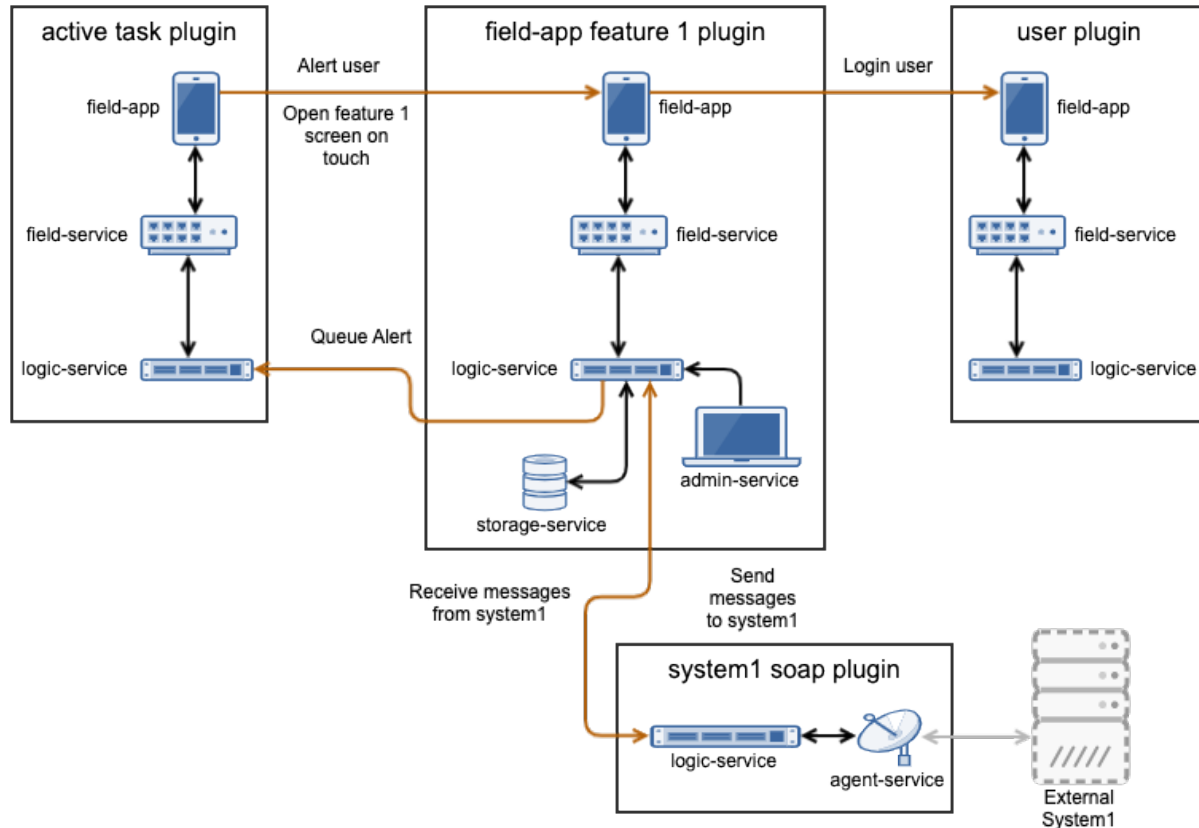
You could create other “User Plugins” with the same exposed plugin API for different backends, and the “Active Task” plugin wouldn’t know the difference.

Stable, exposed APIs make building enterprise applications more manageable.

The next diagram provides an example of how plugins can integrate to each other.

Here are some things of interest :

- The SOAP plugin is implemented to talk specifically to system 1. It handles the burden of implementing the system 1 SOAP interface.
- The SOAP, User and Active Task plugins provide APIs on the logic service that can be multiple feature plugins.
- A feature plugin is just a name we’ve given to the plugin that provides features to the user. It’s no different to any other plugin other than what it does.



2.3.2 One Plugin, One Package

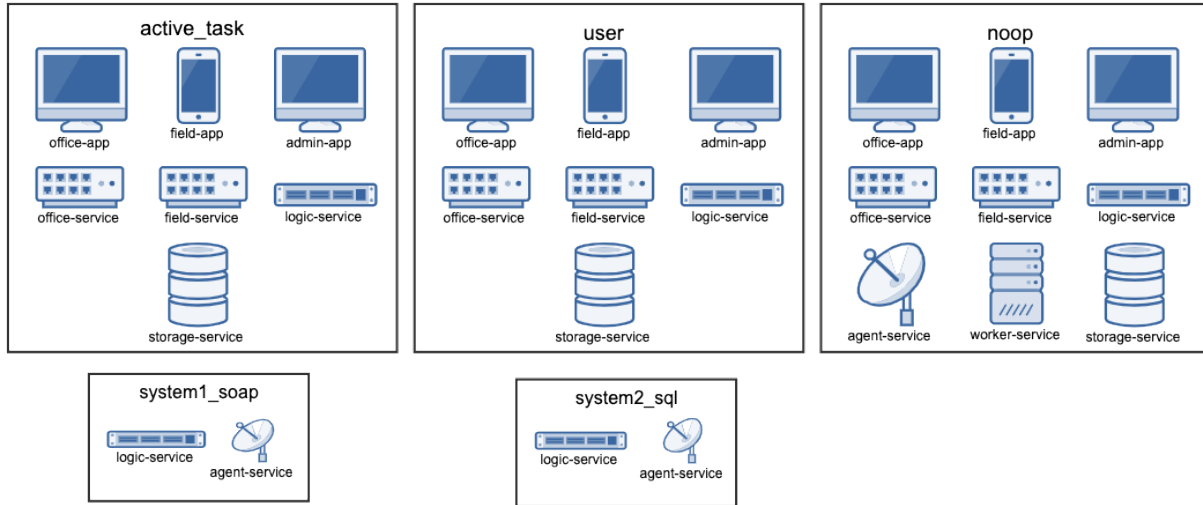
All of the code for one plugin exists within a single python package. This one package is installed on all of the services, even though only part of the plugin will run on each service.

There are multiple entry hooks within the plugin, one for each peek service the plugin chooses to run on.

Each service will start a piece of the plugin, for example: Part of the plugin may run on the logic service, and part of the plugin may run on the agent service.

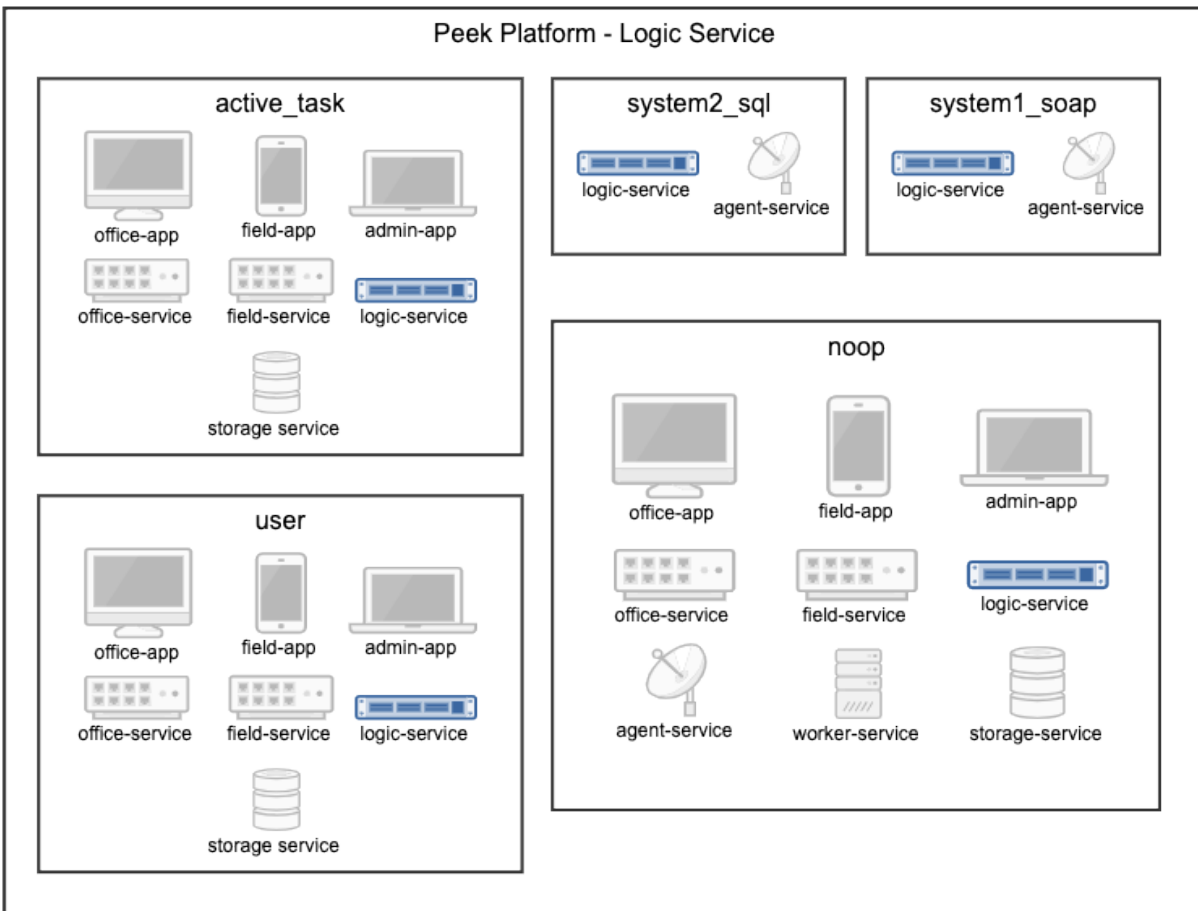
Here are some plugin examples, indicating the services each platform has been designed to run on. Here are some things of interest :

- The User and Active Task plugins don't require the agent or worker services, so they don't have implementation for them.
- All plugins have implementation for the logic service, this is an ideal place for plugins to integrate with each other.



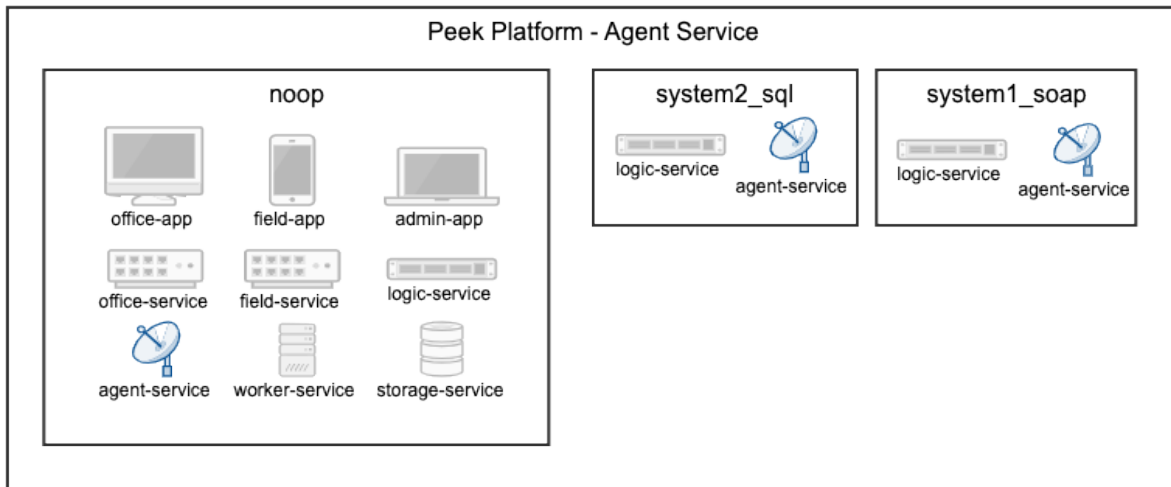
This diagram illustrates how the plugins will run on the logic service.

Each plugins python package is fully installed in the logic services environment. Plugins have entry points for the logic service. The logic service calls this logic service entry hook when it loads each plugin.



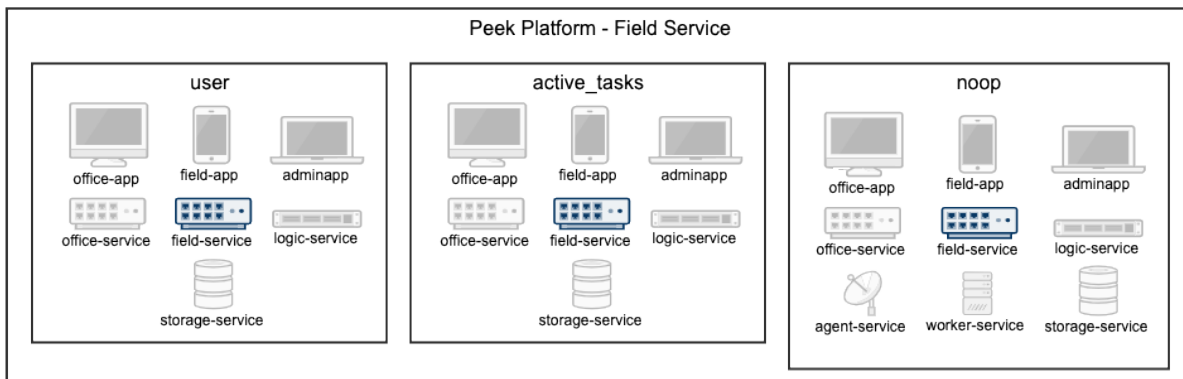
There are only two plugins that require the agent service, so the agent will only load these two. Again, the whole

plugin is installed in the agents python environment.



There are three plugins that require the Office Service, so the Office service will only load these three. Again, the whole plugin is installed in the Office Service python environment.

The field, office, agent, worker and logic services can and run from the one python environment. This is the standard setup for single-server environments.

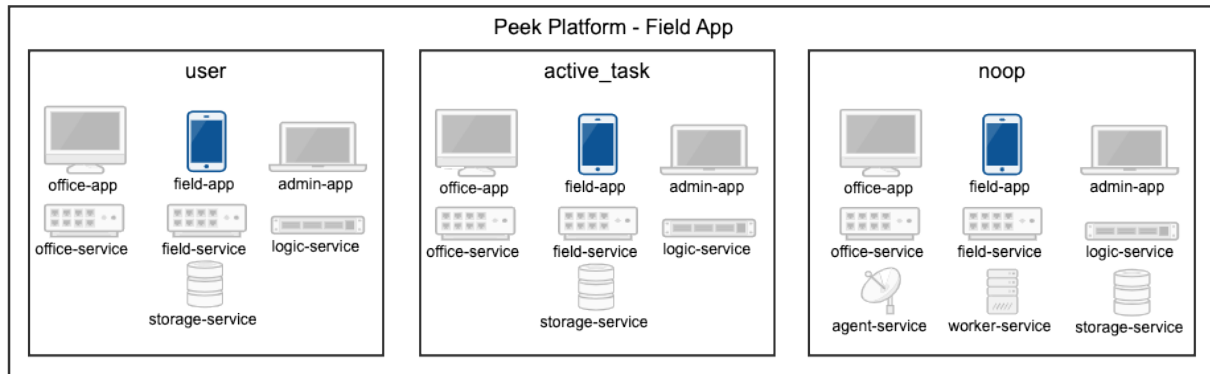


There are three plugins that require the Field App. The Field App is a python package that contains the build skeletons for the web app.

The Field App combines (copies) the files required from each of the plugins into the build environments, and then compiles the web app.

The Field and Logic services prepare and compile the Field and Admin apps, as these are all HTML, SCSS and Typescript.

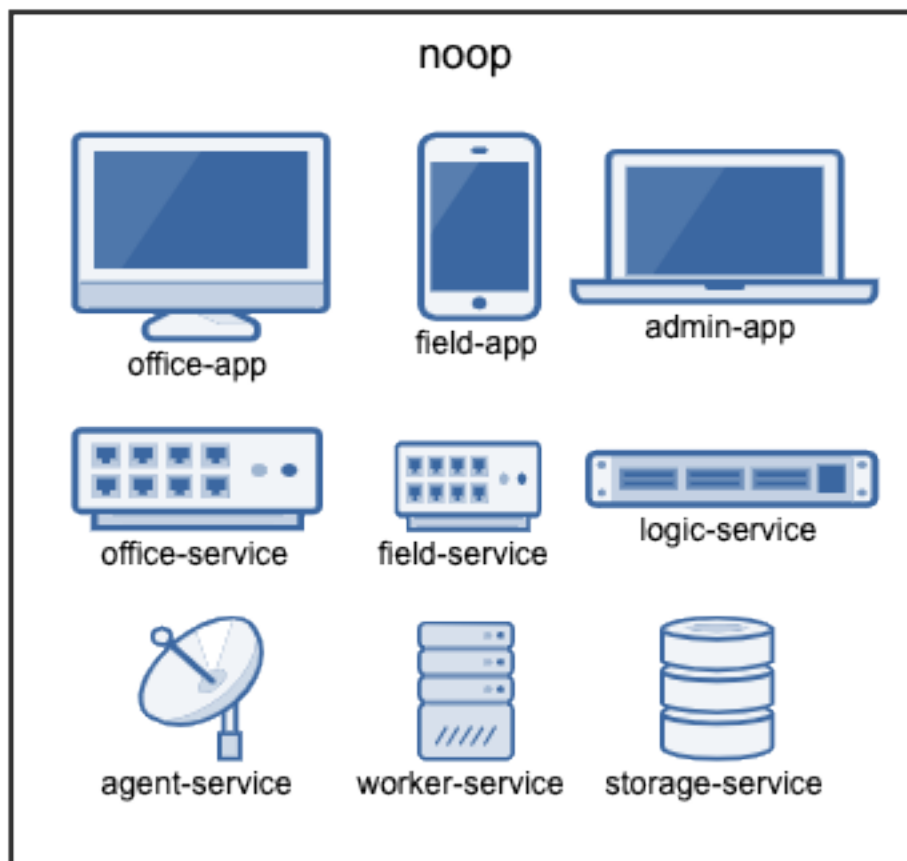
The office/field, and admin interfaces need the office/field, and logic python services to run, so this compile arrangement makes sense.



2.4 Noop Plugin Example

The NOOP plugin is a testing / example plugin.

It's designed to test the basic operations of the platform and runs on every service. All of the code for the plugin is within one python packaged, named "peek-plugin-noop".



The code is available here: [Peek Plugin Noop, on bitbucket](#), It's folder structure looks like this :

- `peek-plugin-noop` (Root project dir, pypi package name)
 - `peek_plugin_noop` (The plugin root, this is the python package)
 - * `_private` (All protected code lives in here) See subfolders below.
 - * `plugin-modules` (Exposed API, `index.ts` will expose public declarations. Plugins can structure the subfolders however they like, this dir is available from `node_modules/@peek/peek_plugin_noop`) See subfolders below.

—
An example contents of the `_private` is described below.

- `_private` (All protected code lives in here)
 - `admin-app` (The admin web based user interface)
 - `admin-assets` (Static assets for the admin web UI)
 - `agent-service` (The code that runs on the agent service)
 - `alembic` (Database schema versioning scripts)
 - `field-service` (The code that runs on the field service)
 - `office-service` (The code that runs on the office service)
 - `office-app` (The office user interface that runs natively and on the mobile/web devices)
 - `office-assets` (Images for the desktop/web)
 - `field-app` (The field user interface that runs natively and on the mobile/web devices)
 - `field-assets` (Images for the mobile/web UI)
 - `logic-service` (The code that runs on the logic service)
 - `:file:storage-service` (SQLAlchemy ORM classes for db access, used by logic and worker)
 - `tuples` (Private data structures)
 - `worker-service` (The parallel processing Celery tasks that are run on the worker)

—
An example contents of the `plugin-modules` is described below.

- `plugin-modules` (Exposed API, `index.ts` will expose public declarations. Plugins can structure the subfolders however they like, this dir is available from `node_modules/@peek/peek_plugin_noop`)
 - `office-app` (Exposed API, `index.ts` exposes office only declarations)
 - `field-app` (Exposed API, `index.ts` exposes field only declarations)
 - `admin-app` (Exposed API, `index.ts` exposes admin only declarations)
 - `_private` (Code only used by this plugin)
 - * `office-app` (Private office declarations)
 - * `field-app` (Private field declarations)
 - * `admin-app` (Private admin declarations)
- `agent-app` (Exposed API, plugins on the agent service use this)
- `field-service` (Exposed API, plugins on the field service use this)
- `office-service` (Exposed API, plugins on the office service use this)

- `logic-service` (Exposed API, plugins on the logic service use this)
 - `tuples` (Exposed Tuples, Tuples on any service use these data structures)
-

Note: Random Fact : Did you know that python can't import packages with hyphens in them?

This document set provides tutorial documentation for the Peek Platform.

3.1 Peek Plugin Tutorial

3.1.1 First Steps

Introduction

The **peek_plugin_base** python package provides all the interfaces used for the Peek Platform and the Plugins to function with each other.

The Platform Plugin API is available here [File *plugin_package.json*](#).

The following sections go on to guide the reader to develop different parts of the plugin and eventually run the plugins in development mode (**ng serve**, **tns run** etc).

Ensure you are well versed with the platform from the [Overview](#) as the following sections build upon that.

The following sections will be useful if you're starting a plugin with out cloning peek_plugin_noop, or if you'd like to learn more about how to code different parts of the plugin.

Check Setup

Important: Windows users must use **bash**

These instructions are cross platform, windows users should use bash from msys, which is easily installable form the windows git installer, see the instructions here, [Setup Msys Git](#).

Check Python

Before running through this procedure, ensure that your PATH variable includes the right virtual environment for the platform you've installed.

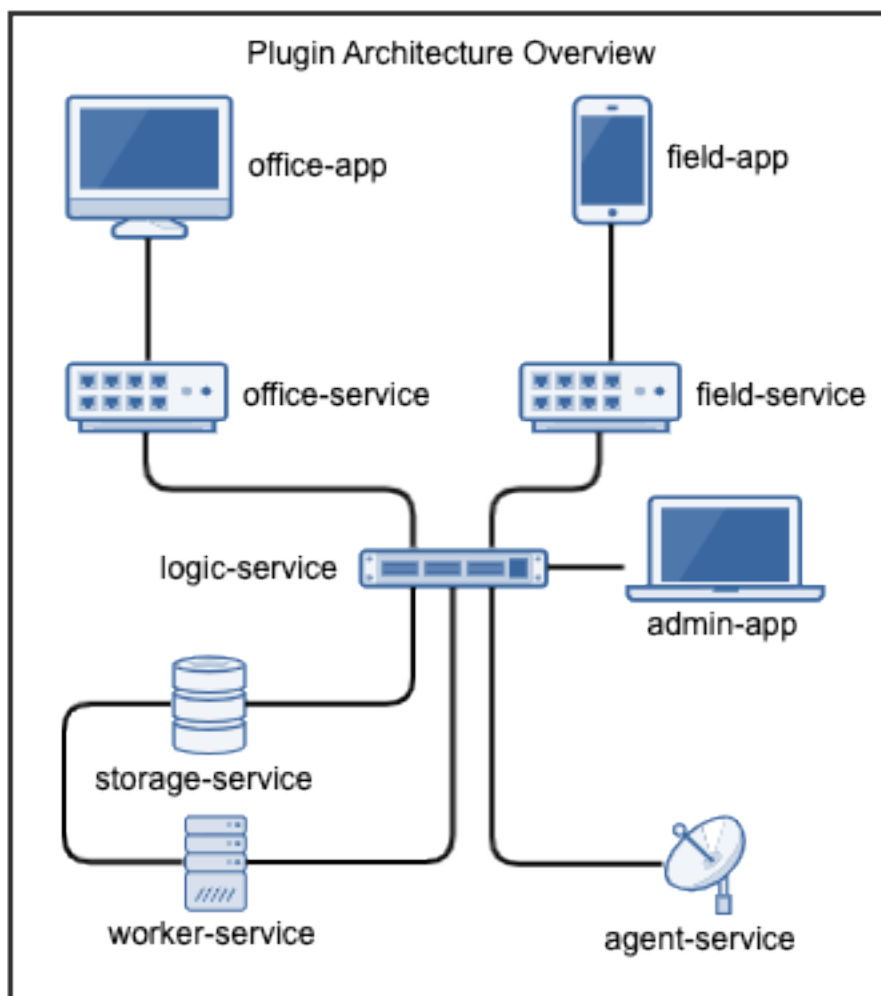
```
which python
```

This should return the location of your virtual environment, usually `~/synerty-peek-V.E.R/bin/python` on Linux or `~/synerty-peek-V.E.R/Script/python` on windows. Where V.E.R is the version number of the platform release, EG 0.2.0

Plugins and the Platform

The Peek Platform services provide places for the Peek Plugins to run. A plugin can chose to run on any service the platform provides.

Here is an architecture diagram for a plugin :



3.1.2 Scaffolding From Scratch

In this section we'll create the basic files we need for a plugin.

Plugin Name peek_plugin_tutorial

We'll finish up with a plugin which we can build a python package for, but it won't run on any services, we'll add that later.

Plugin File Structure

Create Directory peek-plugin-tutorial

peek-plugin-tutorial is the name of the project directory, it could be anything. For consistency, we name it the same as the plugin with hyphens instead of underscores, Python can't import directories with hyphens, so there will be no confusion there.

This directory will contain our plugin package, documentation, build scripts, README, license, etc. These won't be included when the python package is built and deployed.

—

Create the plugin project root directory, and CD to it.

```
peek-plugin-tutorial/
```

Commands:

```
mkdir peek-plugin-tutorial
cd peek-plugin-tutorial
```

Note: Future commands will be run from the plugin project root directory.

Add File .gitignore

The .gitignore file tells the git version control software to ignore certain files in the project. [gitignore](#) - Specifies intentionally untracked files to ignore.

Create .gitignore, and populate it with the following

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# auth generated js and jsmap files
*.js
*.js.map

# Distribution / packaging
.Python
env/
build/
develop-eggs/
*.egg-info
MANIFEST
dist
```

(continues on next page)

(continued from previous page)

```
.idea
.vscode
docs/api_autoapi
```

Add `.editorconfig`

Create the file `.editorconfig`, with the following content:

```
# https://editorconfig.org/

root = true

[*]
indent_style = space
indent_size = 4
insert_final_newline = true
trim_trailing_whitespace = true
end_of_line = lf
charset = utf-8
```

Add Package `peek_plugin_tutorial`

Package `peek_plugin_tutorial` is the root [python package](#). for our plugin.

This package will contain everything that is packaged up and deployed for the Peek Platform to run. This includes:

- The public declarations of the APIs used by other plugins. They are declared using [Python Abstract Base Classes](#).
- Private code that other plugins shouldn't reference.
- Angular2 Components, modules, services and HTML.

Note: Commands will be run from the plugin project root directory, which is `peek-plugin-tutorial`.

Create the `peek_plugin_tutorial` Package. Commands:

```
mkdir -p peek_plugin_tutorial
touch peek_plugin_tutorial/__init__.py
```

Add the version string to the `peek_plugin_tutorial` package.

```
echo "__version__ = '0.0.0'" > peek_plugin_tutorial/__init__.py
```

Note: This version is automatically updated by the `publish.sh` script.

Add Package `_private`

Package `peek_plugin_tutorial._private` will contain the parts of the plugin that won't be exposed/shared for other plugins to use.

Create the `peek_plugin_tutorial._private` Package. Commands:

```
mkdir -p peek_plugin_tutorial/_private
touch peek_plugin_tutorial/_private/__init__.py
```

The structure should now be:

```
peek-plugin-tutorial
├── .gitignore
├── peek_plugin_tutorial
│   ├── __init__.py
│   └── _private
│       └── __init__.py
```

Add File `setup.py`

The `setup.py` file tells the python distribution tools how to create a distributable file for the plugin. [Read more here.](#)

Download `setup.py` from [peek-plugin-noop/setup.py](#)

Modify the options near the top of the file for your plugin. We've modified the following values:

- `py_package_name`
- `description`
- `package_version`

```
#
# Modify these values to fork a new plugin
#
author = "Synerty"
author_email = 'contact@synerty.com'
py_package_name = "peek_plugin_tutorial"
pip_package_name = py_package_name.replace('_', '-')
package_version = '0.0.0'
description = 'Peek Plugin Tutorial - My first enhancement.'

download_url = 'https://bitbucket.org/synerty/%s/get/%s.zip'
download_url %= pip_package_name, package_version
url = 'https://bitbucket.org/synerty/%s' % pip_package_name
```

Add File `publish.sh`

The `publish.sh` file is custom script for building and publishing the plugin that performs the following tasks:

- Updates the version number in the project text files.
- Pushes tags to git

- Copies the built releases to \$RELEASE_DIR if defined
 - Runs setup.py
 - Pushes the release to pypi.python.org
-

Download `publish.sh` from [peek-plugin-noop/publish.sh](#)

Modify the options near the top. We've modified the following:

- PY_PACKAGE

```
#-----  
# Configure package preferences here  
PY_PACKAGE="peek_plugin_tutorial"  
  
# Leave blank not to publish  
# Or select one of the index servers defined in ~/.pypirc  
PYPI_PUBLISH=""
```

Create `publish.settings.sh` with the following content:

```
#!/usr/bin/env bash  
  
PY_PACKAGE="peek_plugin_tutorial"  
PYPI_PUBLISH=""  
  
VER_FILES_TO_COMMIT=""  
  
VER_FILES=""
```

Add File `README.rst`

The file:`README.rst` file is a verbose description of this plugin, it's the file that version control systems, such as BitBucket or GitHub will display when the project is viewed on their sites.

It's ideal to include a great overview about the plugin in this file.

Create a `README`, create a `README.rst` file and populate it.

Here is a suggestion:

```
=====  
Tutorial Plugin 1  
=====
```

This **is** a Peek Plugin, **from the** tutorial.

Add File `plugin_package.json`

The `plugin_package.json` describes the plugin to the Peek Platform. These details include:

- The version
 - The name
 - Which services the plugin needs
 - Additional settings for each service
 - File locations for the Angular applications (admin, office and field)
 - The path of the icon for the plugin,
 - ect.
-

Create the `peek_plugin_tutorial/plugin_package.json` file with the following contents:

```
{
  "plugin": {
    "title": "Tutorial Plugin",
    "packageName": "peek_plugin_tutorial",
    "version": "0.0.0",
    "buildNumber": "#PLUGIN_BUILD#",
    "buildDate": "#BUILD_DATE#",
    "creator": "Synerty Pty Ltd",
    "website": "www.synerty.com"
  },
  "requiresServices": [
  ],
  "admin": {
    "moduleDir": "plugin-module"
  },
  "field": {
    "moduleDir": "plugin-module"
  },
  "office": {
    "moduleDir": "plugin-module"
  }
}
```

Check that your plugin now looks like this:

```
peek-plugin-tutorial
├── peek_plugin_tutorial
│   ├── __init__.py
│   ├── plugin_package.json
│   └── _private
│       └── __init__.py
├── publish.sh
├── README.rst
└── setup.py
```

Add File `PluginNames.py`

The `PluginNames.py` file defines some constants that are used throughout the plugin. More details on where these are used will be later in the documentation.

Since all of the plugin is on the one package, both the part of the plugin running on the logic service and the part of the plugin running on the field or office apps can import this file.

Guaranteeing that there is no mismatch of names when they send data to each other.

Create the `peek_plugin_tutorial/_private/PluginNames.py` file with the following contents:

```
tutorialPluginName = "peek_plugin_tutorial"
tutorialFilt = {"plugin": "peek_plugin_tutorial"}
tutorialTuplePrefix = "peek_plugin_tutorial."
tutorialObservableName = "peek_plugin_tutorial"
tutorialActionProcessorName = "peek_plugin_tutorial"
tutorialTupleOfflineServiceName = "peek_plugin_tutorial"
```

Add Directory `plugin-module/_private`

We now move onto the frontends, and TypeScript.

The `plugin-module/_private` directory will contain code that shouldn't be used outside of this plugin.

The `plugin-module` directory will contain any code that needs to be either:

- Running all the time in the background.
- Shared with other modules.

This directory is sync'd to `node_modules/@peek/peek_plugin_tutorial` on field, admin and office services.

Developers can use some `index.ts` magic to abstract the layout of their directories. An example of importing declaration is as follows:

```
import {tutorialFilt} from "@peek/peek_plugin_tutorial/_private";
```

Create directory `peek_plugin_tutorial/plugin-module/_private`, with command

```
mkdir -p peek_plugin_tutorial/plugin-module/_private
```

Add File `package.json`

The `package.json` file is required to keep NPM from winging, since this directory is linked in under `node_modules/@peek`

Create file `peek_plugin_tutorial/plugin-module/package.json`, with contents

```
{
  "name": "@peek/peek_plugin_tutorial",
  "version": "0.0.0"
}
```

Add File `PluginNames.ts`

The `PluginNames.ts` file defines constants used by this plugin to define, payload filts, tuple names, observable names, etc.

Create file `peek_plugin_tutorial/plugin-module/_private/PluginNames.ts`, with contents

```
export let tutorialFilt = {"plugin": "peek_plugin_tutorial"};
export let tutorialTuplePrefix = "peek_plugin_tutorial.";

export let tutorialObservableName = "peek_plugin_tutorial";
export let tutorialActionProcessorName = "peek_plugin_tutorial";
export let tutorialTupleOfflineServiceName = "peek_plugin_tutorial";

export let tutorialBaseUrl = "peek_plugin_tutorial";
```

Add File `_private/index.ts`

The `_private/index.ts` file defines exports from other files in `_private`.

This lets the code `import tutorialFilt from "@peek/peek_plugin_tutorial/_private";` work instead of `import tutorialFilt from "@peek/peek_plugin_tutorial/_private/PluginNames";`.

It seems trivial at this point, but it becomes more useful as the TypeScript code grows.

Create file `peek_plugin_tutorial/plugin-module/_private/index.ts`, with contents

```
export * from "../PluginNames";
```

Install in Development Mode

Installing the plugin in development mode, links the development directory of the plugin (the directory we create in these instructions) into the python virtual environment.

With this link in place, any python code that wants to use our plugin, is able to import it, and the code run will be the code we're working on.

Install the python plugin package in development mode, run the following:

```
# Check to ensure we're using the right python
which python

python setup.py develop
```

You can test that it's worked with the following python code, run the following in bash:

```
python << EOPY
import peek_plugin_tutorial
import os
print(peek_plugin_tutorial.__version__)
print(os.path.dirname(peek_plugin_tutorial.__file__))
EOPY
```

You now have a basic plugin. In the next section we'll make it run on some services.

3.1.3 Add Documentation

Why does a plugin need documentation? A peek plugin needs documentation to help developers focus on what it needs to do, and allow other developers to use the APIs it shares.

Then it helps Peek admins determine the plugins requirements and if there is a need for it.

Documenting software can be a complicated and a tedious task. There are many things to consider:

- Documentation must be versioned with the code, making sure features match, etc.
- Documentation must be available for each version of the code, your documentation will branch as many times as your code will, 1.0, 1.1, etc
- Documentation must be updated as features are added and changed in the code.

These are a few of the conundrums around the complexity of software documentation. Fortunately there are some fantastic tools around to solve these issues, and you're reading the result of those tools right now.

Sphinx is a tool that makes it easy to create intelligent and beautiful documentation.

Documentation File Structure

The peek-plugin is structured in such a way that the plugin developer can create documentation for 3 different audiences:

- Administrators
- Users
- Developers

Add Admin Documentation

Create directory `peek_plugin_tutorial/doc-admin`:

```
mkdir -p peek_plugin_tutorial/doc-admin
```

Create the file `index.rst`: within the directory `peek_plugin_tutorial/doc-admin` with following content:

```
=====  
Administration  
=====
```

The Peek-Plugin-Tutorial plugin performs the following:

- Point 1
- Point 2

Edit the file `peek_plugin_tutorial/plugin_package.json`:

Add “doc-admin” to the `requiredServices` section so it looks like:

```
"requiredServices": [  
    ...  
    "doc-admin"  
    ...  
]
```

Add the “doc-admin” section after `requiredServices` section:

```
"doc-admin": {  
    "docDir": "doc-admin",  
    "docRst": "index.rst"  
}
```

Ensure your JSON is still valid (Your IDE may help here)

Here is an example:

```
{  
  "plugin": {  
    ...  
  },  
  "requiredServices": [  
    ...  
    "doc-admin"  
    ...  
  ],  
  "doc-admin": {  
    "docDir": "doc-admin",  
    "docRst": "index.rst"  
  }  
}
```

Add User Documentation

Note: These steps are almost identical to the Admin documentation.

Create directory `peek_plugin_tutorial/doc-user`:

```
mkdir -p peek_plugin_tutorial/doc-user
```

Create the file `index.rst`: within the directory `peek_plugin_tutorial/doc-user` with following content:

```
=====
User Guide
=====

This plugin can be used by clicking on the menu icon, etc.
```

Edit the file `peek_plugin_tutorial/plugin_package.json`:

Add “doc-user” to the `requiredServices` section so it looks like:

```
"requiredServices": [
  ...
  "doc-user"
  ...
]
```

Add the “doc-user” section after `requiredServices` section:

```
"doc-user": {
  "docDir": "doc-user",
  "docRst": "index.rst"
}
```

Add Developer Documentation

Note: These steps are almost identical to the Admin documentation.

Create directory `peek_plugin_tutorial/doc-dev`:

```
mkdir -p peek_plugin_tutorial/doc-dev
```

Create the file `index.rst`: within the directory `peek_plugin_tutorial/doc-dev` with following content:

```
=====
Developer
=====

This plugins architecture is as follows <insert images, etc>
```

Edit the file `peek_plugin_tutorial/plugin_package.json`:

Add “doc-dev” to the `requiredServices` section so it looks like:

```
"requiresServices": [  
    ...  
    "doc-dev"  
    ...  
]
```

Add the “doc-dev” section after requiresServices section:

```
"doc-dev": {  
    "docDir": "doc-dev",  
    "docRst": "index.rst",  
    "hasApi": false  
}
```

If your plugin has a public python API, then ensure `hasApi` above is set to `true`.

Check Logic Service Config

The **logic** service builds the **admin** and **dev** documentation.

Edit the `~/peek-logic-service.home/config.json` and ensure the following options are set.

- Ensure `frontend.docBuildEnabled` is set to `true`, with no quotes
- Ensure `frontend.docBuildPrepareEnabled` is set to `true`, with no quotes

Example:

```
{  
    ...  
    "frontend": {  
        ...  
        "docBuildEnabled": true,  
        "docBuildPrepareEnabled": true  
    },  
    ...  
}
```

Check Field Service Config

The **field** service builds the **user** documentation.

Edit the `~/peek-field-service.home/config.json` and ensure the following options are set.

- Ensure `frontend.docBuildEnabled` is set to `true`, with no quotes
- Ensure `frontend.docBuildPrepareEnabled` is set to `true`, with no quotes

Example:

```
{  
    ...  
    "frontend": {  
        ...
```

(continues on next page)

(continued from previous page)

```
        "docBuildEnabled": true,
        "docBuildPrepareEnabled": true
    },
    ...
}
```

CheckOffice Service Config

The **office** service builds the **user** documentation.

Edit the `~/peek-office-service.home/config.json` and ensure the following options are set.

- Ensure `frontend.docBuildEnabled` is set to `true`, with no quotes
- Ensure `frontend.docBuildPrepareEnabled` is set to `true`, with no quotes

Example:

```
{
    ...
    "frontend": {
        ...
        "docBuildEnabled": true,
        "docBuildPrepareEnabled": true
    },
    ...
}
```

Viewing Documentation

The documentation from each peek plugin is loaded into three projects by peek-logic (Admin, Development) and peek-office (User).

The documentation packages are as follows

Administration peek_doc_admin:

Development peek_doc_dev

User peek_doc_user

To view the documentation, you can run `watch_docs.sh`. This will generate the documentation, serve it with a web server and live refresh a web page when a browser is connected to it.

Locate the relevant python project. These instructions will demonstrate with the “Admin” documentation.

Run the following to find the location of `peek_doc_admin`

```
python - <<EOF
import peek_doc_admin
print(peek_doc_admin.__file__)
EOF
```


This will return the following, which you can get the location of `peek_doc_admin` from.

```
peek@_peek ~ % python - <<EOF
import peek_doc_admin
print(peek_doc_admin.__file__)
EOF

/Users/peek/dev-peek/peek-doc-admin/peek_doc_admin/__init__.py
```

Navigate to `peek_doc_admin` from the step above and run the following command:

```
cd /Users/peek/dev-peek/peek-doc-admin/peek_doc_admin
bash watch_docs.sh
```

In your browser, connect to the docs web server that **`watch_docs.sh`** displays at the end of its start.

```
[I 200505 20:51:48 server:296] Serving on http://0.0.0.0:8020
```

Note: The `watch-docs.sh` shell script won't always build a change in the toctree while running. If you update the toctree or modify headings it is good practice to stop `watch-docs.sh`, run `rm -rf dist/*` and restart `watch-docs.sh`.

Note: `version` is the Peek version that is deployed. For example: 2.1.7

Important: Windows users must use **`bash`** and run the commands from the plugin root directory.

For more information on document formatting, please visit [ReStructuredText Cheat Sheet](#).

What Next?

Start developing your own plugins.

3.1.4 Add Logic Service

This section adds the basic files require for the plugin to run on the peek logic service. Create the following files and directories.

Note: Setting up skeleton files for the field, office, worker and agent services, is identical to the logic service, generally replace “logic” with the appropriate service name.

The platform loads the plugins python package, and then calls the appropriate **`peek{logic}EntryHook()`** method on it, if it exists.

The object returned must implement the right interfaces, the platform then calls methods on this object to load, start, stop, unload, etc the plugin.

Logic Service File Structure

Add Package `_private/logic`

This step creates the `_private/logic` python package.

This package will contain the majority of the plugins code that will run on the logic service. Files in this package can be imported with

```
# Example
# To import peek_plugin_tutorial/_private/logic/File.py
from peek_plugin_tutorial._private.logic import File
```

Create directory `peek_plugin_tutorial/_private/logic`

Create an empty package file in the logic directory, `peek_plugin_tutorial/_private/logic/__init__.py`

Commands:

```
mkdir peek_plugin_tutorial/_private/logic
touch peek_plugin_tutorial/_private/logic/__init__.py
```

Add File `LogicEntryHook.py`

This file/class is the entry point for the plugin on the peek logic service. When the peek logic service starts this plugin, it will call the `load()` then the `start()` methods.

Any initialisation and loading that the plugin needs to do to run should be placed in `load()` and `start()` methods.

Important: Ensure what ever is constructed and initialised in the `load()` and `start()` methods, should be deconstructed in the `stop()` and `unload()` methods.

Create the file `peek_plugin_tutorial/_private/logic/LogicEntryHook.py` and populate it with the following contents.

```
import logging

from peek_plugin_base.logic.PluginLogicEntryHookABC import PluginLogicEntryHookABC

logger = logging.getLogger(__name__)

class LogicEntryHook(PluginLogicEntryHookABC):
    def __init__(self, *args, **kwargs):
        """ Constructor """
        # Call the base classes constructor
        PluginLogicEntryHookABC.__init__(self, *args, **kwargs)

        #: Loaded Objects, This is a list of all objects created when we start
        self._loadedObjects = []
```

(continues on next page)

(continued from previous page)

```

def load(self) -> None:
    """ Load

    This will be called when the plugin is loaded, just after the db is migrated.
    Place any custom initialiastion steps here.

    """
    logger.debug("Loaded")

def start(self):
    """ Start

    This will be called to start the plugin.
    Start, means what ever we choose to do here. This includes:

    - Create Controllers

    - Create payload, observable and tuple action handlers.

    """
    logger.debug("Started")

def stop(self):
    """ Stop

    This method is called by the platform to tell the peek app to shutdown and
↪stop everything it's doing
    """
    # Shutdown and dereference all objects we constructed when we started
    while self._loadedObjects:
        self._loadedObjects.pop().shutdown()

    logger.debug("Stopped")

def unload(self):
    """Unload

    This method is called after stop is called, to unload any last resources
    before the PLUGIN is unlinked from the platform

    """
    logger.debug("Unloaded")

```

Edit peek_plugin_tutorial/__init__.py

When the peek logic service loads the plugin, it first calls the **peekLogicEntryHook()** method from the **peek_plugin_tutorial** package.

The **peekLogicEntryHook()** method returns the Class that the peek logic service should create to initialise and start the plugin.

As far as the Peek Platform is concerned, the plugin can be structured how ever it likes internally, as long as it defines these methods in its root python package.

Edit the file `peek_plugin_tutorial/__init__.py`, and add the following:

```
from peek_plugin_base.logic.PluginLogicEntryHookABC import PluginLogicEntryHookABC
from typing import Type

def peekLogicEntryHook() -> Type[PluginLogicEntryHookABC]:
    from ._private.logic.LogicEntryHook import LogicEntryHook
    return LogicEntryHook
```

Edit `plugin_package.json`

These updates to the `plugin_package.json` tell the Peek Platform that we require the “logic” service to run, and additional configuration options we have for that service.

Edit the file `peek_plugin_tutorial/plugin_package.json`:

1. Add “**logic**” to the `requiresServices` section so it looks like

```
"requiresServices": [
    "logic"
]
```

2. Add the **logic** section after **requiresServices** section:

```
"logic": {
}
```

3. Ensure your JSON is still valid (Your IDE may help here)

Here is an example

```
{
  "plugin": {
    ...
  },
  "requiresServices": [
    "logic"
  ],
  "logic": {
  },
  ...
}
```

The plugin should now be ready for the logic service to load.

Running on the Logic Service

File `~/peek-logic-service.home/config.json` is the configuration file for the peek logic service.

Note: This file is created in [Administration](#). Running the Logic Service will also create the file.

Edit `~/peek-logic-service.home/config.json`:

1. Ensure **logging.level** is set to **“DEBUG”**
2. Add **“peek_plugin_tutorial”** to the **plugin.enabled** array

Note: It would be helpful if this is the only plugin enabled at this point.

It should something like this:

```
{
  ...
  "logging": {
    "level": "DEBUG"
  },
  ...
  "plugin": {
    "enabled": [
      "peek_plugin_tutorial"
    ],
    ...
  },
  ...
}
```

You can now run the peek logic service, you should see your plugin load.

```
peek@_peek:~$ run_peek_logic_service
...
DEBUG peek_plugin_tutorial._private.logic.LogicEntryHook:Loaded
DEBUG peek_plugin_tutorial._private.logic.LogicEntryHook:Started
...
```

3.1.5 Add Agent Service

This document is a stripped version of *Add Logic Service*.

Agent Service File Structure

Add Package `_private/agent`

Create directory `peek_plugin_tutorial/_private/agent`

Create an empty package file in the agent directory, `peek_plugin_tutorial/_private/agent/__init__.py`

Commands:

```
mkdir peek_plugin_tutorial/_private/agent
touch peek_plugin_tutorial/_private/agent/__init__.py
```

Add File AgentEntryHook.py

Create the file `peek_plugin_tutorial/_private/agent/AgentEntryHook.py` and populate it with the following contents.

```
import logging

from peek_plugin_base.agent.PluginAgentEntryHookABC import PluginAgentEntryHookABC

logger = logging.getLogger(__name__)

class AgentEntryHook(PluginAgentEntryHookABC):
    def __init__(self, *args, **kwargs):
        """ Constructor """
        # Call the base classes constructor
        PluginAgentEntryHookABC.__init__(self, *args, **kwargs)

        #: Loaded Objects, This is a list of all objects created when we start
        self._loadedObjects = []

    def load(self) -> None:
        """ Load

        This will be called when the plugin is loaded, just after the db is migrated.
        Place any custom initialiaction steps here.

        """
        logger.debug("Loaded")

    def start(self):
        """ Load

        This will be called when the plugin is loaded, just after the db is migrated.
        Place any custom initialiaction steps here.

        """
        logger.debug("Started")

    def stop(self):
        """ Stop

        This method is called by the platform to tell the peek app to shutdown and
        ↪ stop everything it's doing
        """
        # Shutdown and dereference all objects we constructed when we started
        while self._loadedObjects:
            self._loadedObjects.pop().shutdown()

        logger.debug("Stopped")

    def unload(self):
        """Unload

        This method is called after stop is called, to unload any last resources
        before the PLUGIN is unlinked from the platform
        """
```

(continues on next page)

(continued from previous page)

```
"""
logger.debug("Unloaded")
```

Edit peek_plugin_tutorial/__init__.py

Edit the file peek_plugin_tutorial/__init__.py, and add the following:

```
from peek_plugin_base.agent.PluginAgentEntryHookABC import PluginAgentEntryHookABC
from typing import Type

def peekAgentEntryHook() -> Type[PluginAgentEntryHookABC]:
    from ._private.agent.AgentEntryHook import AgentEntryHook
    return AgentEntryHook
```

Edit plugin_package.json

Edit the file peek_plugin_tutorial/plugin_package.json:

1. Add **“agent”** to the requiresServices section so it looks like

```
"requiresServices": [
    "agent"
]
```

2. Add the **agent** section after **requiresServices** section:

```
"agent": {
}
```

3. Ensure your JSON is still valid (Your IDE may help here)

Here is an example

```
{
  "plugin": {
    ...
  },
  "requiresServices": [
    "agent"
  ],
  "agent": {
  }
}
```

The plugin should now be ready for the agent service to load.

Running on the Agent Service

Edit ~/peek-agent-service.home/config.json:

1. Ensure **logging.level** is set to “**DEBUG**”
2. Add “**peek_plugin_tutorial**” to the **plugin.enabled** array

Note: It would be helpful if this is the only plugin enabled at this point.

It should something like this:

```
{
  ...
  "logging": {
    "level": "DEBUG"
  },
  ...
  "plugin": {
    "enabled": [
      "peek_plugin_tutorial"
    ],
    ...
  },
  ...
}
```

Note: This file is created in *Administration*. Running the Agent Service will also create the file.

You can now run the peek agent service, you should see your plugin load.

```
peek@_peek:~$ run_peek_agent_service
...
DEBUG peek_plugin_tutorial._private.agent.AgentEntryHook:Loaded
DEBUG peek_plugin_tutorial._private.agent.AgentEntryHook:Started
...
```

3.1.6 Add Storage Service

The storage service is conceptually a little different to other services in the Peek Platform.

Peek Storage Service connects to a database server, provides each plugin its own schema, and provides much of the boilerplate code required to make this work.

Only two Peek Services are able to access the database, these are the Worker and Logic Services.

The Storage Service schema upgrades are managed by the Logic Service.

Note: The Logic Service must be enabled to use the Storage Service.

Storage Service File Structure

Add Package `_private/storage`

Package `_private/storage` will contain the database ORM classes. These define the schema for the database and are used for data manipulation and retrieval.

Create the `peek_plugin_tutorial._private/storage` Package. Commands:

```
mkdir -p peek_plugin_tutorial/_private/storage
touch peek_plugin_tutorial/_private/storage/__init__.py
```

Add File `DeclarativeBase.py`

The `DeclarativeBase.py` file defines an SQLAlchemy declarative base class. All Table classes inheriting this base class belong together, you can have multiple declarative bases.

See [SQLAlchemy](#) for more details.

In this declarative base, we define a metadata with a schema name for this plugin, `pl_tutorial`.

All the table classes in the plugin will be loaded in this method.

Create a file `peek_plugin_tutorial/_private/storage/DeclarativeBase.py` and populate it with the following contents:

```
from sqlalchemy.ext.declarative import declarative_base

from sqlalchemy.schema import MetaData
from txhttputil.util.ModuleUtil import filterModules

metadata = MetaData(schema="pl_tutorial")
DeclarativeBase = declarative_base(metadata=metadata)


def loadStorageTuples():
    """ Load Storage Tables

    This method should be called from the "load()" method of the agent service, logic_
    ↪service,
    worker service, field service and office service entry hook classes.

    This will register the ORM classes as tuples, allowing them to be serialised and
    deserialized by the vortex.

    """
    for mod in filterModules(__package__, __file__):
        if mod.startswith("Declarative"):
            continue
        __import__(mod, locals(), globals())
```

Add Package `alembic`

Alembic is the database upgrade library Peek uses. The `alembic` package is where the alembic configuration will be kept.

Read more about [Alembic](#) here

Create directory `peek_plugin_tutorial/_private/alembic` Create the empty package file `peek_plugin_tutorial/_private/alembic/__init__.py`

Command:

```
mkdir peek_plugin_tutorial/_private/alembic
touch peek_plugin_tutorial/_private/alembic/__init__.py
```

Add Package versions

The versions package is where the Alembic database upgrade scripts are kept.

Create directory `peek_plugin_tutorial/_private/alembic/versions` Create the empty package file `peek_plugin_tutorial/_private/alembic/versions/__init__.py`

Command:

```
mkdir peek_plugin_tutorial/_private/alembic/versions
touch peek_plugin_tutorial/_private/alembic/versions/__init__.py
```

Add File `env.py`

The `env.py` is loaded by Alembic to get its configuration and environment.

Notice that that `loadStorageTuples()` is called? Alembic needs the table classes loaded to create the version control scripts.

Create a file `peek_plugin_tutorial/_private/alembic/env.py` and populate it with the following contents:

```
from peek_plugin_base.storage.AlembicEnvBase import AlembicEnvBase
from peek_plugin_tutorial._private.storage import DeclarativeBase

DeclarativeBase.loadStorageTuples()

alembicEnv = AlembicEnvBase(DeclarativeBase.metadata)
alembicEnv.run()
```

Add File `script.py.mako`

The `script.py.mako` file is a template that is used by Alembic to create new database version scripts.

Out of interest, Alembic uses [Mako](#) to compile the template into a new script.

Create a file `peek_plugin_tutorial/_private/alembic/script.py.mako` and populate it with the following contents:

```

"""${message}

Peek Plugin Database Migration Script

Revision ID: ${up_revision}
Revises: ${down_revision | comma,n}
Create Date: ${create_date}

"""

# revision identifiers, used by Alembic.
revision = ${repr(up_revision)}
down_revision = ${repr(down_revision)}
branch_labels = ${repr(branch_labels)}
depends_on = ${repr(depends_on)}

from alembic import op
import sqlalchemy as sa
import geoalchemy2
${imports if imports else ""}

def upgrade():
    ${upgrades if upgrades else "pass"}

def downgrade():
    ${downgrades if downgrades else "pass"}

```

Edit File `plugin_package.json`

For more details about the `plugin_package.json`, see [About `plugin\_package.json`](#).

Edit the file `peek_plugin_tutorial/plugin_package.json`:

1. Add “**storage**” to the `requiresServices` section so it looks like

```

"requiresServices": [
    "storage"
]

```

2. Add the **storage** section after **requiresServices** section:

```

"storage": {
    "alembicDir": "_private/alembic"
}

```

3. Ensure your JSON is still valid (Your IDE may help here)

Here is an example

```

{
    ...
    "requiresServices": [
        ...
        "storage"
    ]
}

```

(continues on next page)

(continued from previous page)

```

    ],
    ...
    "storage": {
    }
}

```

Edit File LogicEntryHook.py

The LogicEntryHook.py file needs to be updated to do the following:

- Implement the **PluginLogicEntryHookABC** abstract base class. Including implementing **dbMetadata** property.
- Ensure that the storage Tables are loaded on plugin load.

Edit the file peek_plugin_tutorial/_private/logic/LogicEntryHook.py

1. Add the following import up the top of the file

```

from peek_plugin_tutorial._private.storage import DeclarativeBase
from peek_plugin_tutorial._private.storage.DeclarativeBase import loadStorageTuples
from peek_plugin_base.logic.PluginLogicEntryHookABC import PluginLogicEntryHookABC

```

2. Add **PluginLogicEntryHookABC** to the list of classes “**LogicEntryHook**” inherits

```

class LogicEntryHook(PluginLogicEntryHookABC, PluginLogicEntryHookABC):

```

3. Add the following method from the **load(self):** method

```

def load(self) -> None:
    loadStorageTuples() # <-- Add this line
    logger.debug("Loaded")

```

4. Implement the **dbMetadata(self):** property

```

@property
def dbMetadata(self):
    return DeclarativeBase.metadata

```

When you're finished, You should have a file like this:

```

# Added imports, step 1
from peek_plugin_tutorial._private.storage import DeclarativeBase
from peek_plugin_tutorial._private.storage.DeclarativeBase import loadStorageTuples
from peek_plugin_base.logic.PluginLogicEntryHookABC import \
    PluginLogicEntryHookABC

# Added inherited class, step2
class LogicEntryHook(PluginLogicEntryHookABC, PluginLogicEntryHookABC):

    def load(self) -> None:

```

(continues on next page)

(continued from previous page)

```

    # Added call to loadStorageTables, step 3
    loadStorageTuples()
    logger.debug("Loaded")

    # Added implementation for dbMetadata, step 4
    @property
    def dbMetadata(self):
        return DeclarativeBase.metadata

```

Edit File FieldEntryHook.py

This step applies if your plugin is using the Field Logic service.

The FieldEntryHook.py file needs to be updated to do the following:

- Ensure that the storage service Tables are loaded on plugin load.

Edit the file peek_plugin_tutorial/_private/field/FieldEntryHook.py

1. Add the following import up the top of the file

```

from peek_plugin_tutorial._private.storage.DeclarativeBase import _
↪loadStorageTuples

```

2. Add the following method from the **load(self):** method

```

def load(self) -> None:
    loadStorageTuples() # <-- Add this line
    logger.debug("Loaded")

```

When you're finished, You should have a file like this:

```

# Added imports, step 1
from peek_plugin_tutorial._private.storage.DeclarativeBase import loadStorageTuples

...

def load(self) -> None:
    # Added call to loadStorageTables, step 2
    loadStorageTuples()
    logger.debug("Loaded")

```

Edit File OfficeEntryHook.py

This step applies if your plugin is using the Office Service.

The OfficeEntryHook.py file needs to be updated to do the following:

- Ensure that the storage service Tables are loaded on plugin load.

Edit the file peek_plugin_tutorial/_private/office/OfficeServiceHook.py

1. Add the following import up the top of the file

```
from peek_plugin_tutorial._private.storage.DeclarativeBase import _  
↳loadStorageTuples
```

2. Add the following method from the **load(self):** method

```
def load(self) -> None:  
    loadStorageTuples() # <-- Add this line  
    logger.debug("Loaded")
```

When you're finished, You should have a file like this:

```
# Added imports, step 1  
from peek_plugin_tutorial._private.storage.DeclarativeBase import loadStorageTuples  
  
...  
  
def load(self) -> None:  
    # Added call to loadStorageTables, step 2  
    loadStorageTuples()  
    logger.debug("Loaded")
```

Edit File AgentEntryHook.py

This step applies if you're plugin is using the Agent Service.

Edit file `peek_plugin_tutorial/_private/agent/AgentEntryHook.py` file, apply the same edits from step [Edit File FieldEntryHook.py](#).

Edit File WorkerEntryHook.py

This step applies if you're plugin is using the Worker service.

Edit file `peek_plugin_tutorial/_private/worker/WorkerEntryHook.py` file, apply the same edits from step [Edit File FieldEntryHook.py](#).

Add File alembic.ini

The `alembic.ini` file is the first file Alembic loads, it tells Alembic how to connect to the database and where its "alembic" directory is.

Create a file `peek_plugin_tutorial/_private/alembic.ini` and populate it with the following contents, make sure to update the `sqlalchemy.url` line.

Note: The database connection string is only used when creating database upgrade scripts.

MS Sql Server `mssql+pymssql://peek:PASSWORD@127.0.0.1/peek`

PostgreSQL `postgres://peek:PASSWORD@127.0.0.1/peek`

```
[alembic]
script_location = alembic
sqlalchemy.url = postgresql://peek:PASSWORD@127.0.0.1/peek
```

Finally, run the peek logic service, it should load with out error.

The hard parts done, adding the tables is much easier.

Adding a StringInt Table

This section adds a simple table, For lack of a better idea, lets have a table of strings and Integers.

Add File `StringIntTuple.py`

The `StringIntTuple.py` python file defines a database Table class. This database Table class describes a table in the database.

Most of this is straight from the [SQLAlchemy Object Relational Tutorial](#)

Create the file `peek_plugin_tutorial/_private/storage/StringIntTuple.py` and populate it with the following contents.

```
from sqlalchemy import Column
from sqlalchemy import Integer, String
from vortex.Tuple import Tuple, addTupleType

from peek_plugin_tutorial._private.PluginNames import tutorialTuplePrefix
from peek_plugin_tutorial._private.storage.DeclarativeBase import DeclarativeBase

@addTupleType
class StringIntTuple(Tuple, DeclarativeBase):
    __tupleType__ = tutorialTuplePrefix + 'StringIntTuple'
    __tablename__ = 'StringIntTuple'

    id = Column(Integer, primary_key=True, autoincrement=True)
    string1 = Column(String)
    int1 = Column(Integer)
```

The remainder is from VortexPY, which allows the object to be serialised, and reconstructed as the proper python class. VortexPY is present in these three lines

```
@addTupleType
class StringIntTuple(Tuple, DeclarativeBase):
    __tupleType__ = tutorialTuplePrefix + 'StringIntTuple'
```

Create New Alembic Version

Now we need create a database upgrade script, this allows Peek to automatically upgrade the plugins schema. Peek uses Alembic to handle this.

Read more about [Alembic here](#)

Alembic will load the schema from the database, then load the schema defined by the SQLAlchemy Table classes.

Alembic then works out the differences and create an upgrade script. The upgrade script will modify the database to match the schema defined by the python SQLAlchemy Table classes.

1. Open a **bash** window
2. CD to the `_private` directory of the plugin

```
# Root dir of plugin project
cd peek-plugin-tutorial

# CD to where alembic.ini is
cd peek_plugin_tutorial/_private
```

3. Run the alembic upgrade command.

```
alembic revision --autogenerate -m "Added StringInt Table"
```

it should look like

```
peek@peek:~/project/peek-plugin-tutorial/peek_plugin_tutorial/_private$ alembic_
↪revision --autogenerate -m "Added StringInt Table"
LOAD TABLES
19-Mar-2017 20:59:42 INFO alembic.runtime.migration:Context impl PostgresqlImpl.
19-Mar-2017 20:59:42 INFO alembic.runtime.migration:Will assume transactional DDL.
19-Mar-2017 20:59:42 INFO alembic.autogenerate.compare:Detected added table 'pl_
↪tutorial.StringIntTuple'
/home/peek/cpython-3.5.2/lib/python3.5/site-packages/sqlalchemy/dialects/
↪postgresql/base.py:2705: SAWarning: Skipped unsupported reflection of_
↪expression-based index place_lookup_name_idx
% idx_name)
/home/peek/cpython-3.5.2/lib/python3.5/site-packages/sqlalchemy/dialects/
↪postgresql/base.py:2705: SAWarning: Skipped unsupported reflection of_
↪expression-based index countysub_lookup_name_idx
% idx_name)
/home/peek/cpython-3.5.2/lib/python3.5/site-packages/sqlalchemy/dialects/
↪postgresql/base.py:2705: SAWarning: Skipped unsupported reflection of_
↪expression-based index county_lookup_name_idx
% idx_name)
/home/peek/cpython-3.5.2/lib/python3.5/site-packages/sqlalchemy/dialects/
↪postgresql/base.py:2705: SAWarning: Skipped unsupported reflection of_
↪expression-based index idx_tiger_featnames_lname
% idx_name)
/home/peek/cpython-3.5.2/lib/python3.5/site-packages/sqlalchemy/dialects/
↪postgresql/base.py:2705: SAWarning: Skipped unsupported reflection of_
↪expression-based index idx_tiger_featnames_snd_name
% idx_name)
Generating /home/peek/project/peek-plugin-tutorial/peek_plugin_tutorial/_
↪private/alembic/versions/6c3b8cf5dd77_added_stringint_table.py ... done
```

4. Now check that Alembic has added a new version file in the `peek_plugin_tutorial/_private/alembic/versions` directory.

Tip: You can add any kind of SQL you want to this script, if you want default data, then this is the place to add it.

Now the database needs to be upgraded, run the upgrade script created in the last step, with the following command:

```
alembic upgrade head
```

You should see output similar to:

```
peek@_peek MINGW64 ~/peek-plugin-tutorial/peek_plugin_tutorial/_private
$ alembic upgrade head
21-Mar-2017 02:06:27 INFO alembic.runtime.migration:Context impl PostgresqlImpl.
21-Mar-2017 02:06:27 INFO alembic.runtime.migration:Will assume transactional DDL.
21-Mar-2017 02:06:27 INFO alembic.runtime.migration:Running upgrade -> 0b12f40fadba, ↵
↵Added StringInt Table
21-Mar-2017 02:06:27 DEBUG alembic.runtime.migration:new branch insert 0b12f40fadba
```

Adding a Settings Table

The Noop plugin has special Settings and SettingsProperty tables that is usefully for storing plugin settings.

This section sets this up for the Tutorial plugin. It's roughly the same process used to *Adding a StringInt Table*.

Add File Setting.py

Download the Setting.py file to peek_plugin_tutorial/_private/storage from https://bitbucket.org/synerty/peek-plugin-noop/raw/master/peek_plugin_noop/_private/storage/Setting.py

Edit peek_plugin_tutorial/_private/storage/Setting.py

1. Find **peek_plugin_noop** and replace it with **peek_plugin_tutorial**.
2. Find **noopTuplePrefix** and replace it with **tutorialTuplePrefix**.

Create New Alembic Version

Open a **bash** window, run the alembic upgrade

```
# Root dir of plugin project
cd peek-plugin-tutorial/peek_plugin_tutorial/_private

# Run the alembic command
alembic revision --autogenerate -m "Added Setting Table"
```

Note: Remember to check the file generated, and add it to git.

Run the upgrade script created in the last step with the following command:

```
alembic upgrade head
```

Settings Table Examples

Here is some example code for using the settings table.

Note: This is only example code, you should not leave it in.

Edit the file `peek_plugin_tutorial/_private/logic/LogicEntryHook.py`

Add the following import up the top of the file:

```
from peek_plugin_enmac_events._private.storage.Setting import globalSetting, PROPERTY1
```

To Place this code in the **start ()** : method:

```
# session = self.dbSessionCreator()
#
# # This will retrieve all the settings
# allSettings = globalSetting(session)
# logger.debug(allSettings)
#
# # This will retrieve the value of property1
# value1 = globalSetting(session, key=PROPERTY1)
# logger.debug("value1 = %s" % value1)
#
# # This will set property1
# globalSetting(session, key=PROPERTY1, value="new value 1")
# session.commit()
#
# session.close()
```

3.1.7 Add Admin App

The admin app is the admin user interface. This is known as the “frontend” in web terminology. The backend for the peek admin app is the peek logic service.

In this section we’ll add the root admin page for the plugin.

We only scratch the surface of using Angular, that’s outside the scope of this guide.

See *Developing With The Frontends* to learn more about how Peek pieces together the frontend code from the various plugins.

Admin App File Structure

Add Directory `admin-app`

The `admin-app` directory will contain the plugins the Angular application.

Angular “Lazy Loads” this part of the plugin, meaning it only loads it when the user navigates to the page, and unloads it when it’s finished.

This allows large, single page web applications to be made. Anything related to the user interface should be lazy loaded.

Create directory `peek_plugin_tutorial/_private/admin-app`

Add File `tutorial.component.html`

The `tutorial.component.html` file is the HTML file for the Angular component (`tutorial.component.ts`) we create next.

Create the file `peek_plugin_tutorial/_private/admin-app/tutorial.component.html` and populate it with the following contents.

```
<div class="container">
  <!-- Nav tabs -->
  <ul class="nav nav-tabs" role="tablist">
    <!-- Home Tab -->
    <li role="presentation" class="active">
      <a href="#home" aria-controls="home" role="tab" data-toggle="tab">Home</a>
    </li>
  </ul>

  <!-- Tab panes -->
  <div class="tab-content">
    <!-- Home Tab -->
    <div role="tabpanel" class="tab-pane active" id="home">
      <h1 class="text-center">Tutorial Plugin</h1>
      <p>Angular2 Lazy Loaded Module</p>
      <p>This is the root of the admin app for the Tutorial plugin</p>
    </div>
  </div>
</div>
```

Add File `tutorial.component.ts`

The `tutorial.component.ts` is the Angular Component for the admin page. It's loaded by the default route defined in `tutorial.module.ts`.

See [NgModule](#) for more

Create the file `peek_plugin_tutorial/_private/admin-app/tutorial.component.ts` and populate it with the following contents.

```
import {Component, OnInit} from "@angular/core";

@Component({
  selector: 'tutorial-admin',
  templateUrl: 'tutorial.component.html'
```

(continues on next page)

(continued from previous page)

```
})
export class TutorialComponent implements OnInit {

  ngOnInit() {

  }

}
```

Add File `tutorial.module.ts`

The `tutorial.module.ts` is the main Angular module of the plugin.

This file can describe other routes, that will load other components. This is standard Angular.

See [NgModule](#) for more

Create the file `peek_plugin_tutorial/_private/admin-app/tutorial.module.ts` and populate it with the following contents.

```
import { CommonModule } from "@angular/common";
import { FormsModule } from "@angular/forms";
import { NgModule } from "@angular/core";
import { Routes, RouterModule } from "@angular/router";

// Import our components
import { TutorialComponent } from "../tutorial.component";

// Define the routes for this Angular module
export const pluginRoutes: Routes = [
  {
    path: '',
    pathMatch: 'full',
    component: TutorialComponent
  }
];

// Define the module
@NgModule({
  imports: [
    CommonModule,
    RouterModule.forChild(pluginRoutes),
    FormsModule
  ],
  exports: [],
  providers: [],
  declarations: [TutorialComponent]
})
export class TutorialModule {

}
```

Edit File `plugin_package.json`

Finally, Edit the file `peek_plugin_tutorial/plugin_package.json` to tell the platform that we want to use the `admin-app` service:

1. Add “**admin-app**” to the `requiresServices` section so it looks like

```
"requiresServices": [
  "admin-app"
]
```

2. Add the **admin-app** section after **requiresServices** section:

```
"admin-app": {
  "showHomeLink": true,
  "appDir": "_private/admin-app",
  "appModule": "tutorial.module#TutorialModule"
}
```

3. Ensure your JSON is still valid (Your IDE may help here)

Here is an example

```
{
  ...
  "requiresServices": [
    ...
    "admin"
  ],
  ...
  "admin": {
    ...

    "showHomeLink": true,
    "appDir": "_private/admin-app",
    "appModule": "tutorial.module#TutorialModule"
  }
}
```

Running on the Admin App Service

The Logic Service provides the web service that serves the admin angular application.

The Logic Service takes care of combining all the plugin files into the build directories in the `peek_admin_app` package. We will need to restart Logic Service for it to include our plugin in the admin UI.

See *Developing With The Frontends* for more details.

Check File `~/peek-logic-service.home/config.json`

Check the `~/peek-logic-service.home/config.json` file:

1. Ensure **frontend.webBuildEnabled** is set to **true**, with no quotes
2. Ensure **frontend.webBuildPrepareEnabled** is set to **true**, with no quotes

Note: It would be helpful if this is the only plugin enabled at this point.

Example:

```
{
  ...
  "frontend": {
    ...
    "webBuildEnabled": true,
    "webBuildPrepareEnabled": true
  },
  ...
}
```

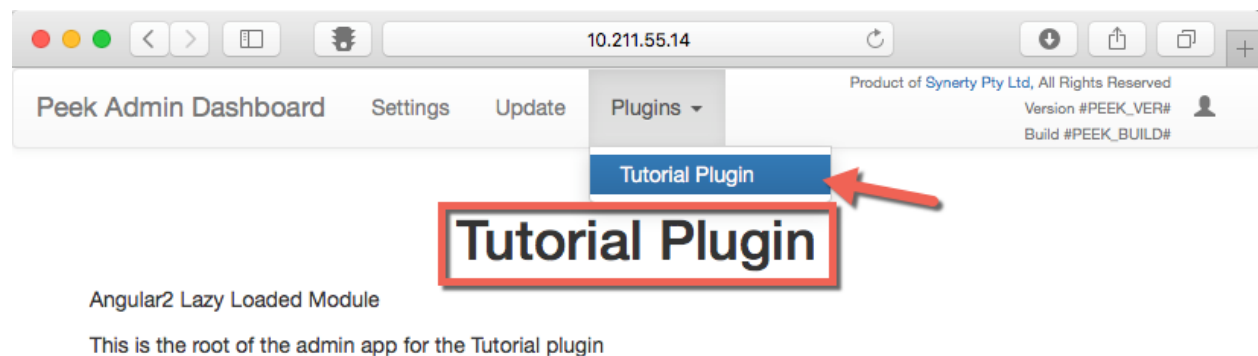
Run run_peek_logic_service

You can now run the peek logic service, you should see your plugin load.

```
peek@peek:~$ run_peek_logic_service
...
INFO peek_platform.frontend.WebBuilder:Rebuilding frontend distribution
...
INFO txhttputil.site.SiteUtil:Peek Admin is alive and listening on http://10.211.55.
↪14:8010
....
```

Now bring up a web browser and navigate to <http://localhost:8010> or the IP mentioned in the output of **run_peek_logic_service**.

If you see this, then congratulations, you’ve just enabled your plugin to use the Peek Platform, Admin App Service.



3.1.8 Add Field App

The field app is for the users. It’s the interface designed for mobile devices.

The field app is known as the “frontend” in web terminology. The backend for the field app is the field service.

The Peek field app is built with an Angular [web build](#).

In this document, we'll add the start of both the field and office builds for the plugin.

We only scratch the surface of using Angular, that's outside the scope of this guide.

See [Developing With The Frontends](#) to learn more about how Peek pieces together the frontend code from the various plugins.

Field App File Structure

Add Directory `field`

The `field` directory will contain the plugins the field Angular application requires.

Angular "Lazy Loads" this part of the plugin, meaning it only loads it when the user navigates to the page, and unloads it when it's finished.

This allows large, single page web applications to be made. Anything related to the user interface should be lazy loaded.

Create directory `peek_plugin_tutorial/_private/field`

Create an empty package file in the field directory, `peek_plugin_tutorial/_private/field/__init__.py`

Commands:

```
mkdir peek_plugin_tutorial/_private/field
touch peek_plugin_tutorial/_private/field/__init__.py
```

Add File `tutorial.component.mweb.html`

The `tutorial.component.mweb.html` file is the web app HTML **view** for the Angular component `tutorial.component.ts`.

This is standard HTML that is compiled by Angular. Angular compiles the HTML, looking for Angular directives, and alters it in place in the browser.

For more information about Angular directives, See:

- [Attribute Directives](#)
 - [Structural Directives](#)
-

Create the file `peek_plugin_tutorial/_private/field/tutorial.component.mweb.html` and populate it with the following contents.

```
<div class="container">
  <h1 class="text-center">Tutorial Plugin</h1>
  <p>Angular2 Lazy Loaded Module</p>
  <p>This is the root of the field app for the Tutorial plugin</p>
</div>
```

Add File `tutorial.component.ts`

The `tutorial.component.ts` is the Angular Component for the field app page. It's loaded by the default route defined in `tutorial.module.ts`.

Note: The one Angular component drives both the Capacitor and Web app views. More on this later.

Create the file `peek_plugin_tutorial/_private/field/tutorial.component.ts` and populate it with the following contents.

```
import {Component} from "@angular/core";

@Component({
  selector: 'plugin-tutorial',
  templateUrl: 'tutorial.component.mweb.html',
  moduleId: module.id
})
export class TutorialComponent {

  constructor() {

  }

}
```

Add File `tutorial.module.ts`

The `tutorial.module.ts` is the main Angular module of the plugin.

This file can describe other routes, that will load other components. This is standard Angular.

[See NgModule](#) for more

Create the file `peek_plugin_tutorial/_private/field/tutorial.module.ts` and populate it with the following contents.

```
import {CommonModule} from "@angular/common";
import {NgModule} from "@angular/core";
import {Routes} from "@angular/router";

import { PeekModuleFactory } from "@synerty/peek-plugin-base-js"

// Import the default route component
import {TutorialComponent} from "../tutorial.component";

// Define the child routes for this plugin
export const pluginRoutes: Routes = [
  {
    path: '',
    pathMatch: 'full',
    component: TutorialComponent
  }
]
```

(continues on next page)

(continued from previous page)

```

    }

};

// Define the root module for this plugin.
// This module is loaded by the lazy loader, what ever this defines is what is
↳started.
// When it first loads, it will look up the routs and then select the component to
↳load.
@NgModule({
  imports: [
    CommonModule,
    PeekModuleFactory.RouterModule,
    PeekModuleFactory.RouterModule.forChild(pluginRoutes),
    ...PeekModuleFactory.FormsModules
  ],
  exports: [],
  providers: [],
  declarations: [TutorialComponent]
})
export class TutorialModule
{
}

```

Download Icon icon.png

The Peek field interface has a home screen with apps on it, this icon will be the tutorial plugins app icon.



Create directory peek_plugin_tutorial/_private/field-assets

Download this plugin app icon [TutorialExampleIcon.png](#) to peek_plugin_tutorial/_private/field-assets/icon.png

Edit File plugin_package.json

Finally, Edit the file peek_plugin_tutorial/plugin_package.json to tell the platform that we want to use the field service:

1. Add “field” to the requiresServices section so it looks like

```

"requiresServices": [
  "field"
]

```

2. Add the **field** section after **requiresServices** section:

```
"field": {
    "showHomeLink": true,
    "appDir": "_private/field-app",
    "appModule": "tutorial.module#TutorialModule",
    "assetDir": "_private/field-assets",
    "icon": "/assets/peek_plugin_tutorial/icon.png"
}
```

3. Ensure your JSON is still valid (Your IDE may help here)

Here is an example

```
{
    ...
    "requiresServices": [
        ...
        "field"
    ],
    ...
    "field": {
        "showHomeLink": true,
        "appDir": "_private/field-app",
        "appModule": "tutorial.module#TutorialModule",
        "assetDir": "_private/field-assets",
        "icon": "/assets/peek_plugin_tutorial/icon.png"
    }
}
```

3.1.9 Add Field Service

This document is a stripped version of *Add Logic Service*.

Create the file `peek_plugin_tutorial/_private/field/FieldEntryHook.py` and populate it with the following contents.

```
import logging

from peek_plugin_base.field.PluginFieldEntryHookABC import PluginFieldEntryHookABC

logger = logging.getLogger(__name__)

class FieldEntryHook(PluginFieldEntryHookABC):
    def __init__(self, *args, **kwargs):
        """ Constructor """
        # Call the base classes constructor
        PluginFieldEntryHookABC.__init__(self, *args, **kwargs)

        #: Loaded Objects, This is a list of all objects created when we start
        self._loadedObjects = []

    def load(self) -> None:
        """ Load

        This will be called when the plugin is loaded, just after the db is migrated.
        """
```

(continues on next page)

(continued from previous page)

```

    Place any custom initialiaistion steps here.

    """
    logger.debug("Loaded")

def start(self):
    """ Load

    This will be called when the plugin is loaded, just after the db is migrated.
    Place any custom initialiaistion steps here.

    """
    logger.debug("Started")

def stop(self):
    """ Stop

    This method is called by the platform to tell the peek app to shutdown and_
↪ stop
    everything it's doing
    """
    # Shutdown and dereference all objects we constructed when we started
    while self._loadedObjects:
        self._loadedObjects.pop().shutdown()

    logger.debug("Stopped")

def unload(self):
    """Unload

    This method is called after stop is called, to unload any last resources
    before the PLUGIN is unlinked from the platform

    """
    logger.debug("Unloaded")

```

Edit the file `peek_plugin_tutorial/__init__.py`, and add the following:

```

from peek_plugin_base.field.PluginFieldEntryHookABC import PluginFieldEntryHookABC
from typing import Type

def peekFieldEntryHook() -> Type[PluginFieldEntryHookABC]:
    from ._private.field.FieldEntryHook import FieldEntryHook
    return FieldEntryHook

```

Edit the file `peek_plugin_tutorial/plugin_package.json`:

1. Add **“field”** to the `requiresServices` section so it looks like

```

"requiresServices": [
    "field",
]

```

2. Add the **field** section after **requiresServices** section:

```
"field": {
},
```

3. Ensure your JSON is still valid (Your IDE may help here)

Here is an example

```
{
  "plugin": {
    ...
  },
  "requiresServices": [
    "field",
  ],
  "field": {
  },
}
```

The plugin should now be ready for the field to load.

Running on the Field Service

Edit `~/peek-field-service.home/config.json`:

1. Ensure **logging.level** is set to “**DEBUG**”
2. Add “**peek_plugin_tutorial**” to the **plugin.enabled** array

Note: It would be helpful if this is the only plugin enabled at this point.

It should something like this:

```
{
  ...
  "logging": {
    "level": "DEBUG"
  },
  ...
  "plugin": {
    "enabled": [
      "peek_plugin_tutorial"
    ],
    ...
  },
  ...
}
```

Note: This file is created in [Administration](#). Running the Field Service will also create the file.

Check File `~/peek-field-service.home/config.json`

Check the `~/peek-field-service.home/config.json` file:

1. Ensure **frontend.webBuildEnabled** is set to **true**, with no quotes
2. Ensure **frontend.webBuildPrepareEnabled** is set to **true**, with no quotes

Note: It would be helpful if this is the only plugin enabled at this point.

Example:

```
{
  ...
  "frontend": {
    ...
    "webBuildEnabled": true,
    "webBuildPrepareEnabled": true
  },
  ...
}
```

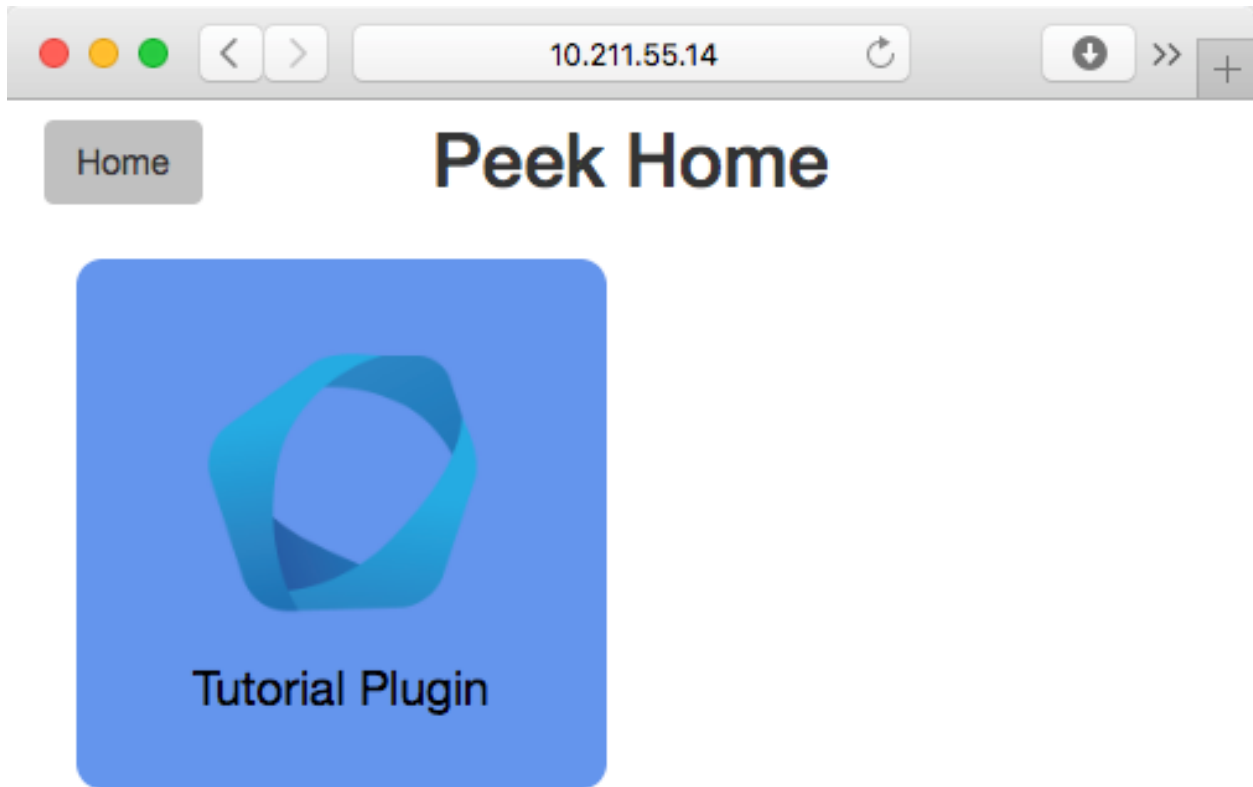
Run run_peek_office_service

You can now run the peek office service, you should see your plugin load.

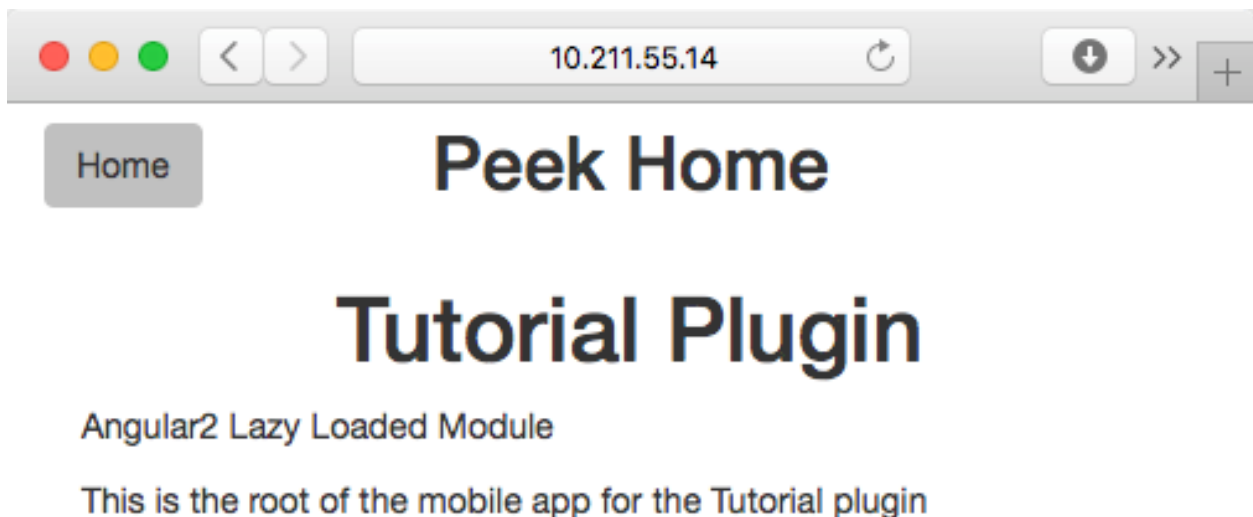
```
peek@_peek:~$ run_peek_office_service
...
DEBUG peek_plugin_tutorial._private.office.OfficeEntryHook:Loaded
DEBUG peek_plugin_tutorial._private.office.OfficeEntryHook:Started
...
INFO peek_platform.frontend.WebBuilder:Rebuilding frontend distribution
...
INFO txhttputil.site.SiteUtil:Peek Office App is alive and listening on http://10.211.
↪55.14:8000
...
```

Now bring up a web browser and navigate to <http://localhost:8000> or the IP mentioned in the output of **run_peek_field_service**.

If you see this, then congratulations, you've just enabled your plugin to use the Peek Platform, Field Service Web App.



Click on the Tutorial app, you should then see your plugins default route component.



3.1.10 Add Office App

The office app is similar to the field app. This document is a stripped version of *Add Field App*.

Office File Structure

Add Package `_private/office`

Create an empty package file in the office directory, `peek_plugin_tutorial/_private/office/__init__.py`

Commands:

```
mkdir peek_plugin_tutorial/_private/office
touch peek_plugin_tutorial/_private/office/__init__.py
```

Add File `tutorial.component.dweb.html`

Create the `peek_plugin_tutorial/_private/office/tutorial.component.dweb.html` with the following contents:

```
<div class="container">
  <h1 class="text-center">Tutorial Plugin</h1>
  <p>Angular2 Lazy Loaded Module</p>
  <p>This is the root of the office app for the Tutorial plugin</p>
</div>
```

Add File `tutorial.component.ts`

Create the file `peek_plugin_tutorial/_private/office/tutorial.component.ts` and populate it with the following contents.

```
import {Component} from "@angular/core";

@Component({
  selector: 'plugin-tutorial',
  templateUrl: 'tutorial.component.dweb.html',
  moduleId: module.id
})
export class TutorialComponent {

  constructor() {

  }

}
```

Create the file `peek_plugin_tutorial/_private/office/tutorial.module.ts` and populate it with the following contents.

```
import {CommonModule} from "@angular/common";
import {NgModule} from "@angular/core";
import {Routes} from "@angular/router";

import { PeekModuleFactory } from "@synerty/peek-plugin-base-js"

// Import the default route component
import {TutorialComponent} from "../tutorial.component";

// Define the child routes for this plugin
export const pluginRoutes: Routes = [
  {
    path: '',
    pathMatch:'full',
    component: TutorialComponent
  }
];

// Define the root module for this plugin.
// This module is loaded by the lazy loader, what ever this defines is what is
// started.
// When it first loads, it will look up the routs and then select the component to
// load.
@NgModule({
  imports: [
    CommonModule,
    PeekModuleFactory.RouterModule,
    PeekModuleFactory.RouterModule.forChild(pluginRoutes),
    ...PeekModuleFactory.FormsModules
  ],
  exports: [],
  providers: [],
  declarations: [TutorialComponent]
})
export class TutorialModule
{
}
```

Download Icon [icon.png](#)

The Peek web interface has a home screen with apps on it, this icon will be the tutorial plugins app icon.



Create directory peek_plugin_tutorial/_private/office-assets

Download this plugin app icon `TutorialExampleIcon.png` to `peek_plugin_tutorial/_private/office-assets/icon.png`

Edit File `plugin_package.json`

Finally, Edit the file `peek_plugin_tutorial/plugin_package.json` to tell the platform that we want to use the office service:

1. Add **office** to the `requiresServices` section so it looks like

```
"requiresServices": [
  "office"
]
```

2. Add the **office** section after `requiresServices` section:

```
"office": {
  "appDir": "_private/office",
  "appModule": "tutorial.module#TutorialModule",
  "assetDir": "_private/office-assets",
  "icon": "/assets/peek_plugin_tutorial/icon.png",
  "showHomeLink": true,
}
```

3. Ensure your JSON is still valid (Your IDE may help here)

Here is an example

```
{
  ...
  "requiresServices": [
    ...
    "office"
  ],
  ...
  "office": {
    "appDir": "_private/office-app",
    "appModule": "tutorial.module#TutorialModule",
    "assetDir": "_private/office-assets",
    "icon": "/assets/peek_plugin_tutorial/icon.png",
    "showHomeLink": true,
  }
}
```

3.1.11 Add Office Service

This document is a stripped version of *Add Logic Service*.

Office Service File Structure

Add File `OfficeEntryHook.py`

Create the file `peek_plugin_tutorial/_private/office/OfficeEntryHook.py` and populate it with the following contents.

```
import logging

from peek_plugin_base.office.PluginOfficeEntryHookABC import PluginOfficeEntryHookABC

logger = logging.getLogger(__name__)

class OfficeEntryHook(PluginOfficeEntryHookABC):
    def __init__(self, *args, **kwargs):
        """ Constructor """
        # Call the base classes constructor
        PluginOfficeEntryHookABC.__init__(self, *args, **kwargs)

        #: Loaded Objects, This is a list of all objects created when we start
        self._loadedObjects = []

    def load(self) -> None:
        """ Load

        This will be called when the plugin is loaded, just after the db is migrated.
        Place any custom initialiastion steps here.

        """
        logger.debug("Loaded")

    def start(self):
        """ Load

        This will be called when the plugin is loaded, just after the db is migrated.
        Place any custom initialiastion steps here.

        """
        logger.debug("Started")

    def stop(self):
        """ Stop

        This method is called by the platform to tell the peek app to shutdown and
↪ stop everything it's doing
        """
        # Shutdown and dereference all objects we constructed when we started
        while self._loadedObjects:
            self._loadedObjects.pop().shutdown()

        logger.debug("Stopped")

    def unload(self):
        """Unload

        This method is called after stop is called, to unload any last resources
        before the PLUGIN is unlinked from the platform

        """
        logger.debug("Unloaded")
```

Edit peek_plugin_tutorial/__init__.py

Edit the file peek_plugin_tutorial/__init__.py, and add the following:

```

from peek_plugin_base.office.PluginOfficeEntryHookABC import PluginOfficeEntryHookABC
from typing import Type

def peekOfficeEntryHook() -> Type[PluginOfficeEntryHookABC]:
    from ._private.office.OfficeEntryHook import OfficeEntryHook
    return OfficeEntryHook

```

Edit plugin_package.json

Edit the file peek_plugin_tutorial/plugin_package.json:

1. Add “office” to the requiresServices section so it looks like

```

"requiresServices": [
    "office",
]

```

2. Add the **office** section after **requiresServices** section:

```

"office": {
}

```

3. Ensure your JSON is still valid (Your IDE may help here)

Here is an example

```

{
    "plugin": {
        ...
    },
    "requiresServices": [
        "office",
    ],
    "office": {
    }
}

```

The plugin should now be ready for the office to load.

Running on the Office Service

Edit ~/peek-office-service.home/config.json:

1. Ensure **logging.level** is set to “**DEBUG**”
2. Add “**peek_plugin_tutorial**” to the **plugin.enabled** array

Note: It would be helpful if this is the only plugin enabled at this point.

It should something like this:

```
{
  ...
  "logging": {
    "level": "DEBUG"
  },
  ...
  "plugin": {
    "enabled": [
      "peek_plugin_tutorial"
    ],
    ...
  },
  ...
}
```

Note: This file is created in *Administration*. Running the Office Service will also create the file.

Run run_peek_office_service

Run the peek office service

```
peek@_peek:~$ run_peek_office_service
```

you should see your plugin load.

```
peek@_peek:~$ run_peek_office_service
...
DEBUG peek_plugin_tutorial._private.office.OfficeEntryHook:Loaded
DEBUG peek_plugin_tutorial._private.office.OfficeEntryHook:Started
...
INFO txhttputil.site.SiteUtil:Peek Office Site is alive and listening on http://0.0.0.0.
↪0:8002
...
```

Now bring up a web browser and navigate to <http://localhost:8002> or the IP mentioned in the output of **run_peek_office_service**.

3.1.12 Add Tuples

In this document, define tuples in Python and TypeScript. A Tuple is a defined class in TypeScript (javascript) or Python.

These are not to be confused with the `tuple` python built in type.

What are it's purposes:

1. We can work with first class objects `t1.string1`, VS dicts of attributes `t1["string1"]`.
2. We can add additional methods to the Tuple classes that would not otherwise be available, EG `t1.formattedStringInt()`
3. Defining Tuples simplifies sending data between serv ices via the vortex, If a Tuple object is sent on one end, it will be a Tuple object when it's deserailised on the other end.

Important: It's important to import all the tuples when the plugin is loaded on each Peek python service (worker, field, office, logic and agent).

The plugin loading code will throw errors if one of our Tuples is imported first by another plugin and not by us.

Objective

In this procedure we'll do the following:

1. Create a Tuple in Python and register it.
2. Create a Tuple in TypeScript and register it.
3. Create a StringIntTuple in TypeScript and register it.

Tuples File Structure

Add Package `_private.tuples`

The `_private.tuples` python package will contain the private python Tuples.

Create the `peek_plugin_tutorial/_private/tuples` package, with the commands

```
mkdir peek_plugin_tutorial/_private/tuples
touch peek_plugin_tutorial/_private/tuples/__init__.py
```

Add File `TutorialTuple.py`

The `TutorialTuple.py` defines a simple class that we use to work with data. This is serialisable by the Vortex.

Create the file `peek_plugin_tutorial/_private/tuples/TutorialTuple.py` and populate it with the following contents.

```
from vortex.Tuple import Tuple, addTupleType, TupleField

from peek_plugin_tutorial._private.PluginNames import tutorialTuplePrefix

@addTupleType
class TutorialTuple(Tuple):
    """ Tutorial Tuple

    This tuple is a create example of defining classes to work with our data.
    """
    __tupleType__ = tutorialTuplePrefix + 'TutorialTuple'

    #: Description of datel
    dict1 = TupleField(defaultValue=dict)
```

(continues on next page)

(continued from previous page)

```
#: Description of date1
array1 = TupleField(defaultValue=list)

#: Description of date1
date1 = TupleField()
```

Edit File `_private/tuples/__init__.py`

In this step, we add a setup method on the tuples package, this setup method then loads all the handlers needed for the backend.

Edit file `peek_plugin_tutorial/_private/tuples/__init__.py` Add the following:

```
from txhttputil.util.ModuleUtil import filterModules

def loadPrivateTuples():
    """ Load Private Tuples

    In this method, we load the private tuples.
    This registers them so the Vortex can reconstructed them from
    serialised data.

    """
    for mod in filterModules(__name__, __file__):
        __import__(mod, locals(), globals())
```

Add Package `tuples`

The `tuples` python package will contain the public python `Tuples`. The tuples which our plugin wants to share with other plugins.

We won't define any public tuples here, but we'll set it up.

See more at [Add Plugin Python API](#).

Create the `peek_plugin_tutorial/tuples` package, with the commands

```
mkdir peek_plugin_tutorial/tuples
touch peek_plugin_tutorial/tuples/__init__.py
```

Edit File `tuples/__init__.py`

In this step, we add a setup method on the tuples package, this setup method then loads all the handlers needed for the backend.

Edit file `peek_plugin_tutorial/tuples/__init__.py` Add the following:

```

from txhttputil.util.ModuleUtil import filterModules

def loadPublicTuples():
    """ Load Public Tuples

    In this method, we load the public tuples.
    This registers them so the Vortex can reconstructed them from
    serialised data.

    """

    for mod in filterModules(__name__, __file__):
        __import__(mod, locals(), globals())

```

Edit File LogicEntryHook.py

Now, we need to load all our Tuples when the plugin is loaded, for every service. To do this, we call the methods we've added to the tuple packages above.

Edit file `peek_plugin_tutorial/_private/logic/LogicEntryHook.py`:

1. Add this import up the top of the file

```

from peek_plugin_tutorial._private.tuples import loadPrivateTuples
from peek_plugin_tutorial.tuples import loadPublicTuples

```

2. Add this line after the docstring in the `load()` method

```

loadPrivateTuples()
loadPublicTuples()

```

The method should now look similar to this

```

def load(self):
    ...
    loadStorageTuples() # This line was added in the "Add Storage" guide
    loadPrivateTuples()
    loadPublicTuples()
    logger.debug("Loaded")

```

Note: If you see a message like this in the log: `Tuple type |%s| not registered within this program`. The above steps haven't been completed properly and there is a problem with the tuple loading in the peek services.

Edit File FieldEntryHook.py

This step applies if you're plugin is using the Field Service.

Note: This service was add earlier in this tutorial, see [Add Field Service](#)

Edit file `peek_plugin_tutorial/_private/field/FieldEntryHook.py` file, apply the same edits from step [Edit File LogicEntryHook.py](#).

Edit File `OfficeEntryHook.py`

This step applies if you're plugin is using the Field Service.

Note: This service was add earlier in this tutorial, see [Add Office Service](#)

Edit file `peek_plugin_tutorial/_private/office/OfficeEntryHook.py` file, apply the same edits from step [Edit File LogicEntryHook.py](#).

Edit File `AgentEntryHook.py`

This step applies if you're plugin is using the Agent service.

Note: This service was add earlier in this tutorial, see [Add Agent Service](#)

Edit file `peek_plugin_tutorial/_private/agent/AgentEntryHook.py` file, apply the same edits from step [Edit File LogicEntryHook.py](#).

Edit File `WorkerEntryHook.py`

This step applies if you're plugin is using the Worker service.

Note: This service is added in this tutorial, see [Add Worker Service](#)

Edit file `peek_plugin_tutorial/_private/worker/WorkerEntryHook.py` file, apply the same edits from step [Edit File LogicEntryHook.py](#).

Test Python Services

At this point all the python services should run, you won't see any differences but it's a good idea to run them all and check there are no issues.

Tuples Frontends and TypeScript

We now move onto the frontends, and TypeScript.

Add Directory `plugin-module/_private/tuples`

The `plugin-module/_private/tuples` directory will contain our example tuple, written in TypeScript.

Our exampled tuple will be importable with:

```
import {TutorialTuple} from "@peek/peek_plugin_tutorial";
```


Create directory `peek_plugin_tutorial/plugin-module/_private/tuples`, with command

```
mkdir -p peek_plugin_tutorial/plugin-module/_private/tuples
```

Edit File `plugin_package.json`

Edit the file `plugin_package.json` to include reference to **plugin-module** inside the block **field** and **admin**. Your file should look similar to the below:

```
{
  "admin": {
    ...
    "moduleDir": "plugin-module"
  },
  "field": {
    ...
    "moduleDir": "plugin-module"
  },
  "office": {
    ...
    "moduleDir": "plugin-module"
  },
  ...
}
```

Add File `TutorialTuple.ts`

The `TutorialTuple.ts` file defines a TypeScript class for our `TutorialTuple` Tuple.

Create file `peek_plugin_tutorial/plugin-module/_private/tuples/TutorialTuple.ts`, with contents

```
import {addTupleType, Tuple} from "@synerty/vortexjs";
import {tutorialTuplePrefix} from "../PluginNames";

@addTupleType
export class TutorialTuple extends Tuple {
  public static readonly tupleName = tutorialTuplePrefix + "TutorialTuple";

  // Description of date1
  date1 : Date;

  // Description of array1
  array1 : any[];

  // Description of date1
  date1 : Date;
```

(continues on next page)

(continued from previous page)

```
    constructor() {  
        super(TutorialTuple.tupleName)  
    }  
}
```

Add File `StringIntTuple.ts`

The `StringIntTuple.ts` file defines the TypeScript Tuple for the hybrid Tuple/SQL Declarative that represents `StringIntTuple`.

Create file `peek_plugin_tutorial/plugin-module/_private/tuples/StringIntTuple.ts`, with contents

```
import {addTupleType, Tuple} from "@synerty/vortexjs";  
import {tutorialTuplePrefix} from "../PluginNames";  
  
@addTupleType  
export class StringIntTuple extends Tuple {  
    public static readonly tupleName = tutorialTuplePrefix + "StringIntTuple";  
  
    // Description of datel  
    id : number;  
  
    // Description of string1  
    string1 : string;  
  
    // Description of int1  
    int1 : number;  
  
    constructor() {  
        super(StringIntTuple.tupleName)  
    }  
}
```

Add File `SettingPropertyTuple.ts`

The `SettingPropertyTuple.ts` file defines the TypeScript Tuple for the hybrid Tuple/SQL Declarative that represents `SettingPropertyTuple`.

The `SettingProperty` storage table is the in the `storage/Settings.py` file, It's the table that stores the key/value pairs.

Create file `peek_plugin_tutorial/plugin-module/_private/tuples/SettingPropertyTuple.ts`, with contents

```
import {addTupleType, Tuple} from "@synerty/vortexjs";  
import {tutorialTuplePrefix} from "../PluginNames";
```

(continues on next page)

(continued from previous page)

```

@addTupleType
export class SettingPropertyTuple extends Tuple {
  // The tuple name here should end in "Tuple" as well, but it doesn't, as it's a
  ↪table
  public static readonly tupleName = tutorialTuplePrefix + "SettingProperty";

  id: number;
  settingId: number;
  key: string;
  type: string;

  int_value: number;
  char_value: string;
  boolean_value: boolean;

  constructor() {
    super(SettingPropertyTuple.tupleName)
  }
}

```

Edit File `_private/index.ts`

The `_private/index.ts` file will re-export the `Tuple` in a more standard way. Developers won't need to know the exact path of the file.

Edit file `peek_plugin_tutorial/plugin-module/_private/index.ts`, Append the line:

```

export {TutorialTuple} from "./tuples/TutorialTuple";
export {StringIntTuple} from "./tuples/StringIntTuple";
export {SettingPropertyTuple} from "./tuples/SettingPropertyTuple";

```

This document is complete.

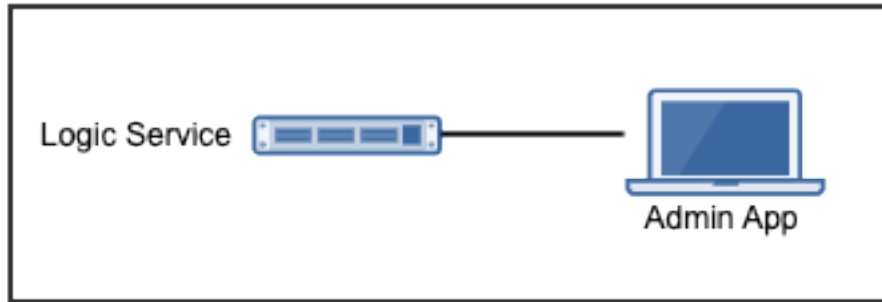
3.1.13 Add Tuple Loader

Outline

In this document, we'll use the `TupleLoader` from `VortexJS` to load and update some settings from the tables created in [Adding a StringInt Table](#) and tuples created in [Add Tuples](#).

The Admin and Logic services talk to each other via a `Vortex`, this is the name given to the transport layer of `VortexJS` and `VortexPY`.

A plugin developer could choose to use standard HTTP requests with JSON, however, the `Vortex` maintains a persistent connection unless it's shutdown.



This document modifies both the logic and admin parts of the plugin.

Advantages

1. Easily edit table data.

Disadvantages

1. Not suitable for multiple users.

Logic Service Scaffold

This section sets up the non specific files needed when we add the Tuple Load Handlers.

Add Package `admin_backend`

The `admin_backend` python package will contain the classes that provide data sources to the Admin web app.

Create the `peek_plugin_tutorial/_private/logic/admin_backend` package, with the commands

```
mkdir peek_plugin_tutorial/_private/logic/admin_backend
touch peek_plugin_tutorial/_private/logic/admin_backend/__init__.py
```

Edit File `admin_backend/__init__.py`

In this step, we add a setup method on the `admin_backend` package, this setup method then loads all the handlers needed for the backend.

This just helps sectionalise the code a bit.

The `makeAdminBackendHandlers` method is a generator because we use `yield`. We can yield more items after the first one, the calling will get an iterable return.

Edit file `peek_plugin_tutorial/_private/logic/admin_backend/__init__.py` Add the following:

```
def makeAdminBackendHandlers(dbSessionCreator):
    pass
```

Edit File LogicEntryHook.py

Now, we need to create and destroy our `admin_backend` handlers when the Logic Service starts the plugin.

If you look at `self._loadedObjects`, you'll see that the `stop()` method shuts down all objects we add to this array. So adding to this array serves two purposes

1. It keeps a reference to the object, ensuring it isn't garbage collected when the `start()` method ends.
2. It ensures all the objects are properly shutdown. In our case, this means it stops listening for payloads.

Edit file `peek_plugin_tutorial/_private/logic/LogicEntryHook.py`:

1. Add this import up the top of the file

```
from .admin_backend import makeAdminBackendHandlers
```

2. Add this line after the docstring in the `start()` method

```
self._loadedObjects.extend(makeAdminBackendHandlers(self.dbSessionCreator))
```

The method should now look similar to this

```
def start(self):
    """ Load

    This will be called when the plugin is loaded, just after the db is migrated.
    Place any custom initialiaiation steps here.

    """
    self._loadedObjects.extend(makeAdminBackendHandlers(self.dbSessionCreator))
    logger.debug("Started")
```

Test Python Services

The backend changes are complete, please run **run_peek_logic_service** to ensure that there are no problems here.

StringInt Logic Service

Add the handler that will listen to the StringInt tuple loader.

Add File StringIntTableHandler.py

The `StringIntTableHandler.py` listens for payload from the Admin service (frontend) These payloads are delivered by the vortex.

When the `OrmCrudHandler` class in the Logic services receives the payloads from the `TupleLoader` in the Admin frontend, it creates, reads, updates or deletes (CRUD) data in the the database.

Create the file `peek_plugin_tutorial/_private/admin_backend/StringIntTableHandler.py` and populate it with the following contents.

```
import logging

from peek_plugin_tutorial._private.PluginNames import tutorialFilt
from peek_plugin_tutorial._private.storage.StringIntTuple import StringIntTuple

from vortex.sqla_orm.OrmCrudHandler import OrmCrudHandler

logger = logging.getLogger(__name__)

# This dict matches the definition in the Admin angular app.
filtKey = {"key": "admin.Edit.StringIntTuple"}
filtKey.update(tutorialFilt)

# This is the CRUD handler
class __CrudHandler(OrmCrudHandler):
    pass

    # If we only wanted to edit a subset of the data, this is how it's done
    # def createDeclarative(self, session, payloadFilt):
    #     lookupName = payloadFilt["lookupName"]
    #     return (session.query(StringIntTuple)
    #             .filter(StringIntTuple.lookupName == lookupName)
    #             .all())

# This method creates an instance of the handler class.
def makeStringIntTableHandler(dbSessionCreator):
    handler = __CrudHandler(dbSessionCreator, StringIntTuple,
                           filtKey, retrieveAll=True)

    logger.debug("Started")
    return handler
```

Edit File `admin_backend/__init__.py`

In this step, we add a setup method on the `admin_backend` package, this setup method then loads all the handlers needed for the backend.

This just helps sectionalise the code a bit.

The `makeAdminBackendHandlers` method is a generator because we use `yield`. We can yield more items after the first one, the calling will get an iterable return.

Edit file `peek_plugin_tutorial/_private/logic/admin_backend/__init__.py`

1. Add the following python import to the top fo the file

```
from .StringIntTableHandler import makeStringIntTableHandler
```

#. Find the method `def makeAdminBackendHandlers(dbSessionCreator)`: Add the following line to it

```
yield makeStringIntTableHandler(dbSessionCreator)
```

StringInt Admin Service

This section adds the tuple loader support in for the StringInt test tuple. these changes are in TypeScript and run in Angular / The frontend.

Add Directory `edit-string-int-table`

The `edit-string-int-table` directory will contain the view and controller that allows us to edit data in the admin app.

Create the `peek_plugin_tutorial/_private/admin-app/edit-string-int-table` directory, with the command

```
mkdir peek_plugin_tutorial/_private/admin-app/edit-string-int-table
```

Add File `edit.component.html`

The `edit.component.html` file is the HTML file for the Angular component (`edit.component.ts`) we create next.

This view will display the data, allow us to edit it and save it.

Create the file `peek_plugin_tutorial/_private/admin-app/edit-string-int-table/edit.component.html` and populate it with the following contents.

```
<div class="panel panel-default">
  <div class="panel-heading">Edit String Ints
    <div class="btn-toolbar pull-right">
      <div class="btn-group">
        <div class="btn btn-default btn-sm" (click)='save()' '>
          Save
        </div>
        <div class="btn btn-default btn-sm" (click)='resetClicked()' '>
          Reset
        </div>
        <div class="btn btn-default btn-sm" (click)='addRow()' '>
          Add
        </div>
      </div>
    </div>
  </div>
  <div class="panel-body">
    <table class="table">
      <tr>
        <th>String 1</th>
        <th>Int 1</th>
        <th></th>
```

(continues on next page)

(continued from previous page)

```

    </tr>
    <tr *ngFor="let item of items">
      <td>
        <input [(ngModel)]="item.string1"
              class="form-control input-sm"
              type="text"/>
      </td>
      <td>
        <input [(ngModel)]="item.int1"
              class="form-control input-sm"
              type="number"/>
      </td>
      <td>
        <div class="btn btn-default" (click)='removeRow(item) '>
          <span class="glyphicon glyphicon-minus" aria-hidden="true"></
↪span>
        </div>
      </td>
    </tr>
  </table>
</div>
</div>

```

There are two buttons in this HTML that are related to the TupleLoader, these call methods on the loader, `loader.save(items)`, `loader.load()`.

Add File `edit.component.ts`

The `edit.component.ts` is the Angular Component for the new edit page.

In this component:

1. We inherit from `NgLifecycleEvents`, this provides a little automatic unsubscription magic for VortexJS
2. We define the `filt`, this is a dict that is used by payloads to describe where payloads should be routed to on the other end.
3. We ask Angular to inject the Vortex services we need, this is in the constructor.
4. We get the `VortexService` to create a new `TupleLoader`.
5. We subscribe to the data from the `TupleLoader`.

Create the file `peek_plugin_tutorial/_private/admin-app/edit-string-int-table/edit.component.ts` and populate it with the following contents.

```

import {Component, OnInit} from "@angular/core";
import { BalloonMsgService, NgLifecycleEvents } from "@synerty/peek-plugin-base-js"
import {
  extend,
  VortexService,
  TupleLoader
} from "@synerty/vortexjs";
import {StringIntTuple,
  tutorialFilt
} from "@peek/peek_plugin_tutorial/_private";

```

(continues on next page)

(continued from previous page)

```

@Component ({
  selector: 'pl-tutorial-edit-string-int',
  templateUrl: './edit.component.html'
})
export class EditStringIntComponent extends NgLifecycleEvents {
  // This must match the dict defined in the admin_backend handler
  private readonly filt = {
    "key": "admin.Edit.StringIntTuple"
  };

  items: StringIntTuple[] = [];
  itemsToDelete: StringIntTuple[] = [];

  loader: TupleLoader;

  constructor(private balloonMsg: BalloonMsgService,
               vortexService: VortexService) {
    super();

    this.loader = vortexService.createTupleLoader(this,
      () => {
        let filt = extend({}, this.filt, tutorialFilt);
        // If we wanted to filter the data we get, we could add this
        // filt["lookupName"] = 'lookupType';
        return filt;
      });

    this.loader.observable
      .subscribe((tuples:StringIntTuple[]) => {
        this.items = tuples;
        this.itemsToDelete = [];
      });
  }

  addRow() {
    let t = new StringIntTuple();
    // Add any values needed for this list here, EG, for a lookup list you might
    ↪add:
    // t.lookupName = this.lookupName;
    this.items.push(t);
  }

  removeRow(item) {
    if (item.id != null)
      this.itemsToDelete.push(item);

    let index: number = this.items.indexOf(item);
    if (index != -1) {
      this.items.splice(index, 1);
    }
  }

  save() {
    let itemsToDelete = this.itemsToDelete;

```

(continues on next page)

(continued from previous page)

```

        this.loader.save(this.items)
            .then(() => {
                if (itemsToDelete.length != 0) {
                    return this.loader.del(itemsToDelete);
                }
            })
            .then(() => this.balloonMsg.showSuccess("Save Successful"))
            .catch(e => this.balloonMsg.showError(e));
    }

    resetClicked() {
        this.loader.load()
            .then(() => this.balloonMsg.showSuccess("Reset Successful"))
            .catch(e => this.balloonMsg.showError(e));
    }
}

```

Edit File tutorial.component.html

Update the tutorial.component.html to insert the new EditStringIntComponent component into the HTML.

Edit the file peek_plugin_tutorial/_private/admin-app/tutorial.component.html:

1. Find the tag and insert the following before that line:

```

<!-- Edit String Int Tab -->
<li role="presentation">
    <a href="#editStringInt" aria-controls="editStringInt" role="tab"
        data-toggle="tab">Edit String Int</a>
</li>

```

2. Find the <div class="tab-content"> tag and insert the following after the line it:

```

<!-- Edit String Int Tab -->
<div role="tabpanel" class="tab-pane" id="editStringInt">
    <pl-tutorial-edit-string-int></pl-tutorial-edit-string-int>
</div>

```

Edit File tutorial.module.ts

Edit the tutorial.module.ts Angular Module to import the EditStringIntComponent component.

Edit the peek_plugin_tutorial/_private/admin-app/tutorial.module.ts:

1. Add this import statement with the imports at the top of the file:

```

import {EditStringIntComponent} from "../edit-string-int-table/edit.component";

```

2. Add EditStringIntComponent to the declarations array, EG:

```
declarations: [TutorialComponent, EditStringIntComponent]
```

Test StringInt Tuple Loader

Restart the Logic Service, so that it rebuilds the Admin Angular Web app.

Navigate your browser to the admin page, select plugins, and then select the “Edit String Int” tab.

Settings Logic Service

Add the handler that will listen to the StringInt tuple loader.

Add File SettingPropertyHandler.py

The SettingPropertyHandler.py listens for payload from the Admin service (frontend) These payloads are delivered by the vortex.

Create the file peek_plugin_tutorial/_private/admin_backend/SettingPropertyHandler.py and populate it with the following contents.

```
import logging
from vortex.sqla_orm.OrmCrudHandler import OrmCrudHandler

from peek_plugin_tutorial._private.PluginNames import tutorialFilt
from peek_plugin_tutorial._private.storage.Setting import SettingProperty, ↵
↳globalSetting

logger = logging.getLogger(__name__)

# This dict matches the definition in the Admin angular app.
filtKey = {"key": "admin.Edit.SettingProperty"}
filtKey.update(tutorialFilt)

# This is the CRUD handler
class __CrudHandler(OrmCrudHandler):
    # The UI only edits the global settings
    # You could get more complicated and have the UI edit different groups of ↵
    ↳settings.
    def createDeclarative(self, session, payloadFilt):
        return [p for p in globalSetting(session).propertyObjects]

# This method creates an instance of the handler class.
def makeSettingPropertyHandler(dbSessionCreator):
    handler = __CrudHandler(dbSessionCreator, SettingProperty,
                            filtKey, retrieveAll=True)

    logger.debug("Started")
    return handler
```

Edit File `admin_backend/__init__.py`

In this step, we add the new handler to the `makeAdminBackendHandlers` function, this will start them when the plugin loads.

Edit file `peek_plugin_tutorial/_private/logic/admin_backend/__init__.py`

1. Add the following python import to the top fo the file

```
from .SettingPropertyHandler import makeSettingPropertyHandler
```

#. Find the method `def makeAdminBackendHandlers(dbSessionCreator)` : Add the following line to it

```
yield makeSettingPropertyHandler(dbSessionCreator)
```

Settings Admin Service

This section adds the tuple loader support in for the `SettingProperty` tuples. These changes are in TypeScript and run in Angular / The frontend.

Add Directory `edit-setting-table`

The `edit-setting-table` directory will contain the view and controller that allows us to edit settings in the admin app.

Create the `peek_plugin_tutorial/_private/admin-app/edit-setting-table` directory, with the command

```
mkdir peek_plugin_tutorial/_private/admin-app/edit-setting-table
```

Add File `edit.component.html`

The `edit.component.html` file is the HTML file for the Angular component (`edit.component.ts`) we create next.

This view will display the data, allow us to edit it and save it.

Create the file `peek_plugin_tutorial/_private/admin-app/edit-setting-table/edit.component.html` and populate it with the following contents.

```
<div class="panel panel-default">
  <div class="panel-body">
    <form autocomplete="off" novalidate>
      <table class="table">
        <tr>
          <th>Setting</th>
          <th>Value</th>
```

(continues on next page)

(continued from previous page)

```

        </tr>
        <tr *ngFor="let item of items">
            <td>{{item.key}}</td>
            <td *ngIf="item.type == 'boolean' ">
                <Button class="btn"
                    [class.btn-success]="item.boolean_value"
                    [class.btn-danger]="!item.boolean_value"
                    (click)="item.boolean_value = ! item.boolean_value">
                    {{item.boolean_value ? "True" : "False"}}
                </Button>
            </td>
            <td *ngIf="item.type == 'integer' ">
                <input [(ngModel)]="item.int_value"
                    [name]="item.key"
                    type="number"
                    step="1"
                    class="form-control input-sm"/>
            </td>
            <td *ngIf="item.key.endsWith('pass') && item.type == 'string' ">
                <input [(ngModel)]="item.char_value"
                    [name]="item.key"
                    type="password"
                    class="form-control input-sm"/>
            </td>
            <td *ngIf="!item.key.endsWith('pass') && item.type == 'string' ">
                <input [(ngModel)]="item.char_value"
                    [name]="item.key"
                    class="form-control input-sm"/>
            </td>
        </tr>
    </table>

    <div class="btn-toolbar">
        <div class="btn-group">
            <div class="btn btn-default" (click)='saveClicked()'>
                Save
            </div>
            <div class="btn btn-default" (click)='resetClicked()'>
                Reset
            </div>
        </div>
    </div>
</form>
</div>
</div>

```

There are two buttons in this HTML that are related to the TupleLoader, these call methods on the loader, `loader.save(items)`, `loader.load()`.

Add File `edit.component.ts`

The `edit.component.ts` is the Angular Component for the new edit settings page.

Create the file `peek_plugin_tutorial/_private/admin-app/edit-setting-table/edit.component.ts` and populate it with the following contents.

```
import {Component} from "@angular/core";
import { BalloonMsgService, NgLifecycleEvents } from "@synerty/peek-plugin-base-js"
import {
    extend,
    TupleLoader,
    VortexService
} from "@synerty/vortexjs";
import {SettingPropertyTuple, tutorialFilt} from "@peek/peek_plugin_tutorial/_private
↪";

@Component ({
    selector: 'pl-tutorial-edit-setting',
    templateUrl: './edit.component.html'
})
export class EditSettingComponent extends NgLifecycleEvents {
    // This must match the dict defined in the admin_backend handler
    private readonly filt = {
        "key": "admin.Edit.SettingProperty"
    };

    items: SettingPropertyTuple[] = [];

    loader: TupleLoader;

    constructor(private balloonMsg: BalloonMsgService,
                 vortexService: VortexService) {
        super();

        this.loader = vortexService.createTupleLoader(this,
            () => extend({}, this.filt, tutorialFilt));

        this.loader.observable
            .subscribe((tuples:SettingPropertyTuple[]) => this.items = tuples);
    }

    saveClicked() {
        this.loader.save()
            .then(() => this.balloonMsg.showSuccess("Save Successful"))
            .catch(e => this.balloonMsg.showError(e));
    }

    resetClicked() {
        this.loader.load()
            .then(() => this.balloonMsg.showSuccess("Reset Successful"))
            .catch(e => this.balloonMsg.showError(e));
    }
}
```

Edit File tutorial.component.html

Update the tutorial.component.html to insert the new EditSettingComponent component into the HTML.

Edit the file peek_plugin_tutorial/_private/admin-app/tutorial.component.html:

1. Find the `</ul>` tag and insert the following before that line:

```
<!-- Edit Settings Tab -->
<li role="presentation">
  <a href="#editSetting" aria-controls="editSetting" role="tab"
    data-toggle="tab">Edit Settings</a>
</li>
```

2. Find the `<div class="tab-content">` tag and insert the following after the line it:

```
<!-- Edit Settings Tab -->
<div role="tabpanel" class="tab-pane" id="editSetting">
  <pl-tutorial-edit-setting></pl-tutorial-edit-setting>
</div>
```

Edit File `tutorial.module.ts`

Edit the `tutorial.module.ts` Angular Module to import the `EditSettingComponent` component.

Edit the `peek_plugin_tutorial/_private/admin-app/tutorial.module.ts`:

1. Add this import statement with the imports at the top of the file:

```
import {EditSettingComponent} from "../edit-setting-table/edit.component";
```

2. Add `EditSettingComponent` to the declarations array, EG:

```
declarations: [TutorialComponent, EditStringIntComponent, EditSettingComponent]
```

Test Settings Tuple Loader

Restart the Logic Service, so that it rebuilds the Admin Angular Web app.

Navigate your browser to the admin page, select plugins, and then select the “Edit Settings” tab.

3.1.14 Add Offline Storage

Outline

The Offline Storage is used by the Field and Office services. It provides an easy way to save and load tuples in the devices

This data can be accessed offline, or loaded before the Field or Office service has responded to a request for data.

In this document, we setup a provider for the Angular Service.

Field Service

Edit File `tutorial.module.ts`

Edit the `tutorial.module.ts` Angular module for the tutorial plugin to add the provider entry for the storage service.

Edit the file `peek_plugin_tutorial/_private/field-app/tutorial.module.ts`:

1. Add the following imports:

```
// Import the required classes from VortexJS
import {
  TupleOfflineStorageNameService,
  TupleOfflineStorageService
} from "@synerty/vortexjs";

// Import the names we need for the
import {
  tutorialTupleOfflineServiceName
} from "@peek/peek_plugin_tutorial/_private";
```

2. After the imports, add this function

```
export function tupleOfflineStorageNameServiceFactory() {
  return new TupleOfflineStorageNameService(tutorialTupleOfflineServiceName);
}
```

3. Finally, add this snippet to the providers array in the @NgModule decorator

```
TupleOfflineStorageService, {
  provide: TupleOfflineStorageNameService,
  useFactory: tupleOfflineStorageNameServiceFactory
},
```

It should look similar to the following:

```
...

import {
  TupleOfflineStorageNameService,
  TupleOfflineStorageService
} from "@synerty/vortexjs";
import {
  tutorialTupleOfflineServiceName
} from "@peek/peek_plugin_tutorial/_private";

...

export function tupleOfflineStorageNameServiceFactory() {
  return new TupleOfflineStorageNameService(tutorialTupleOfflineServiceName);
}

@NgModule({
  ...
  providers: [
    ...
    TupleOfflineStorageService, {
      provide: TupleOfflineStorageNameService,
      useFactory: tupleOfflineStorageNameServiceFactory
    },
    ...
  ],
})
```

(continues on next page)

(continued from previous page)

```
    ]
  })
  export class TutorialModule {
  }
```

Complete.

The tutorial plugin is now setup to use the TupleOffline service. This service is used by TupleActionPushOfflineService and TupleDataOfflineObserverService services.

A developer can use the TupleOfflineStorageService service if they wish but that's outside the scope of this tutorial.

3.1.15 Add Observables

Outline

In this document, we setup the Tuple Observable from VortexJS. The Field and Office services use this to request and receive data updates from the Logic service.

We'll use the term "devices" interchangeably with Field/Office.

This is a one directional data flow once the initial request has been made, the Server will send updates to the Field/Office without the Field/Office services polling for it.

In the example setup, the Field/Office Services proxy Observable requests/responses between the Logic Service and Field/Office devices. The Proxy on the Field/Office service is aware of all the Field/Office App devices that want to observe the data, the Logic Service only knows that the Field/Office Service is observing the data.

Note: The Field/Office devices don't and can't talk directly to the Logic Service.

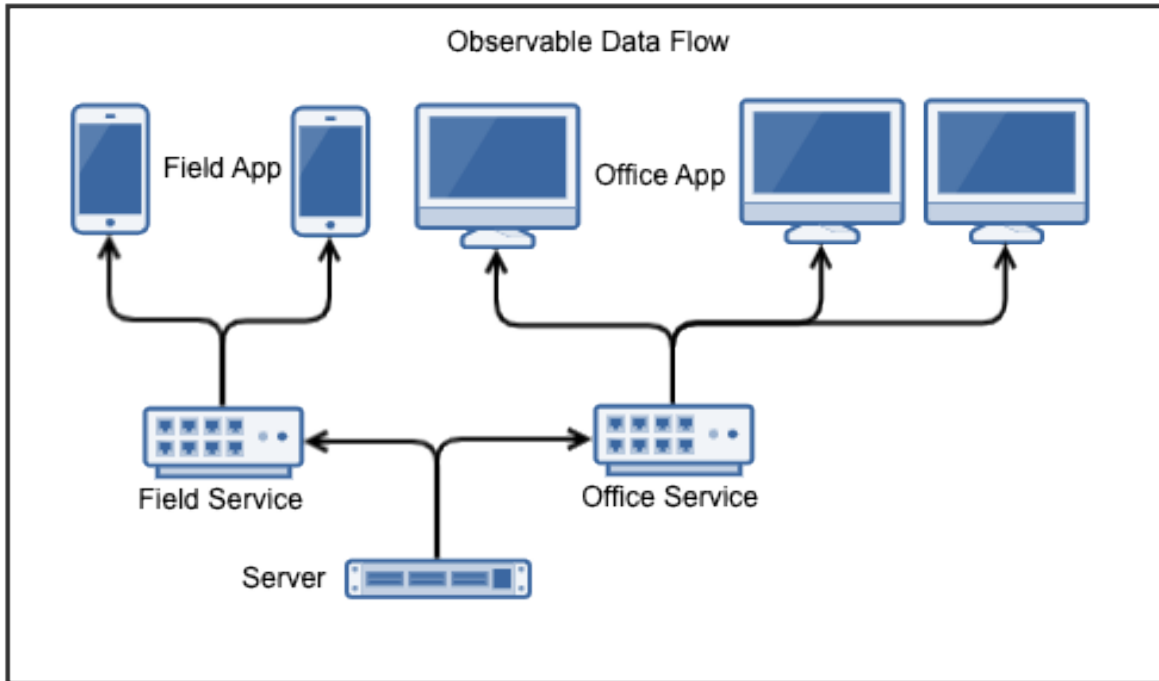
The TupleDataObservableHandler class provides the "observable" functionality. It receives request for data by being sent a TupleSelector. The TupleSelector describes the Tuple type and some conditions of the data the observer wants.

Advantages

1. Instant and efficient data updates, data immediately sent to the devices without the devices congesting bandwidth with polls.

Disadvantages

1. There is no support for updates.



Objective

In this document, our plugin will observe updates made to the table created in [Adding a StringInt Table](#) via the admin web app.

This is the order:

1. Add the Observable scaffolding for the project.
2. Add the Logic Service side Tuple Provider
3. Tell the Admin TupleLoader to notifyDeviceInfo the Observable when it makes updates.
4. Add a new Angular component to observe and display the data.

Server Service Setup

Add Package `tuple_providers`

The `tuple_providers` python package will contain the classes that generate tuple data to send via the observable.

Create the `peek_plugin_tutorial/_private/server/tuple_providers` package, with the commands

```
mkdir peek_plugin_tutorial/_private/server/tuple_providers
touch peek_plugin_tutorial/_private/server/tuple_providers/__init__.py
```

Add File TupleDataObservable.py

The TupleDataObservable.py creates the Observable, registers the tuple providers (they implement TuplesProviderABC)

TupleProviders know how to get the Tuples.

Create the file peek_plugin_tutorial/_private/server/TupleDataObservable.py and populate it with the following contents.

```
from vortex.handler.TupleDataObservableHandler import TupleDataObservableHandler

from peek_plugin_tutorial._private.PluginNames import tutorialFilt
from peek_plugin_tutorial._private.PluginNames import tutorialObservableName

def makeTupleDataObservableHandler(ormSessionCreator):
    """ Make Tuple Data Observable Handler

    This method creates the observable object, registers the tuple providers and then
    returns it.

    :param ormSessionCreator: A function that returns a SQLAlchemy session when called

    :return: An instance of :code:`TupleDataObservableHandler`

    """
    tupleObservable = TupleDataObservableHandler(
        observableName=tutorialObservableName,
        additionalFilt=tutorialFilt)

    # Register TupleProviders here

    return tupleObservable
```

Edit File LogicEntryHook.py

We need to update LogicEntryHook.py, it will initialise the observable object when the Plugin is started.

Edit the file peek_plugin_tutorial/_private/logic/LogicEntryHook.py:

1. Add this import at the top of the file with the other imports:

```
from .TupleDataObservable import makeTupleDataObservableHandler
```

2. Add this line after the docstring in the start() method:

```
tupleObservable = makeTupleDataObservableHandler(self.dbSessionCreator)
self._loadedObjects.append(tupleObservable)
```

The observable for the Logic Service is setup now. We'll add a TupleProvider later.

Field Service Setup

Add File DeviceTupleDataObservableProxy.py

The DeviceTupleDataObservableProxy.py creates the Observable Proxy. This class is responsible for proxying observable data between the devices and the Logic Service.

It reduces the load on the logic service, providing the ability to create more client (Field/Office, for example) services to scale Peek out for more users, or speed up responsiveness for remote locations.

TupleProviders know how to get the Tuples.

Create the file peek_plugin_tutorial/_private/field/DeviceTupleDataObservableProxy.py and populate it with the following contents.

```
from peek_plugin_base.PeekVortexUtil import peekServerName
from peek_plugin_tutorial._private.PluginNames import tutorialFilt
from peek_plugin_tutorial._private.PluginNames import tutorialObservableName
from vortex.handler.TupleDataObservableProxyHandler import _
↳ TupleDataObservableProxyHandler

def makeDeviceTupleDataObservableProxy():
    return TupleDataObservableProxyHandler(observableName=tutorialObservableName,
                                           proxyToVortexName=peekServerName,
                                           additionalFilt=tutorialFilt)
```

Edit File FieldEntryHook.py

We need to update FieldEntryHook.py, it will initialise the observable proxy object when the Plugin is started.

Edit the file peek_plugin_tutorial/_private/field/FieldEntryHook.py:

1. Add this import at the top of the file with the other imports:

```
from .DeviceTupleDataObservableProxy import makeDeviceTupleDataObservableProxy
```

2. Add this line after the docstring in the start() method:

```
self._loadedObjects.append(makeDeviceTupleDataObservableProxy())
```

Field App Setup

Now we need to edit the Angular module in the field-app and add the providers:

Edit File tutorial.module.ts

Edit the tutorial.module.ts Angular module for the tutorial plugin to add the provider entry for the Observer service.

Edit the file peek_plugin_tutorial/_private/-app/tutorial.module.ts:

1. Add the following imports:

```
// Import the required classes from VortexJS
import {
  TupleDataObservableNameService,
  TupleDataObserverService,
  TupleDataOfflineObserverService
} from "@synerty/vortexjs";

// Import the names we need for the
import {
  tutorialObservableName,
  tutorialFilt
} from "@peek/peek_plugin_tutorial/_private";
```

2. After the imports, add this function

```
export function tupleDataObservableNameServiceFactory() {
  return new TupleDataObservableNameService(
    tutorialObservableName, tutorialFilt);
}
```

3. Finally, add this snippet to the providers array in the @NgModule decorator

```
TupleDataObserverService, TupleDataOfflineObserverService, {
  provide: TupleDataObservableNameService,
  useFactory: tupleDataObservableNameServiceFactory
},
```

It should look similar to the following:

```
...

import {
  TupleDataObserverService,
  TupleDataObservableNameService,
  TupleDataOfflineObserverService,
} from "@synerty/vortexjs";

import {
  tutorialObservableName,
  tutorialFilt
} from "@peek/peek_plugin_tutorial/_private";

...

export function tupleDataObservableNameServiceFactory() {
  return new TupleDataObservableNameService(
    tutorialObservableName, tutorialFilt);
}

@NgModule({
  ...
  providers: [
    ...
    TupleDataObserverService, TupleDataOfflineObserverService, {
      provide: TupleDataObservableNameService,
```

(continues on next page)

(continued from previous page)

```
        useFactory:tupleDataObservableNameServiceFactory
    },
    ...
]
}))
export class TutorialModule {
}
```

At this point, all of the observable setup is done. It's much easier to work with the observable code from here on.

Add Tuple Provider

Add File `StringIntTupleProvider.py`

The Observable will be sent a `TupleSelector` that describes the data the sender wants to subscribe to.

Tuple Selectors have two attributes :

1. A name, the name/type of the Type
2. And a selector, this allows the subscriber to observe a filtered set of tuples.

The `StringIntTupleProvider.py` loads data from the database, converts it to a `VortexMsg` and returns it.

A `VortexMsg` is a bytes python type. It's a serialised and compressed payload. A Payload is the Vortex transport container.

Create the file `peek_plugin_tutorial/_private/server/tuple_providers/StringIntTupleProvider.py` and populate it with the following contents.

```
from txhttputil.util.DeferUtil import deferToThreadWrap
from typing import Union

from twisted.internet.defer import Deferred

from vortex.Payload import Payload
from vortex.TupleSelector import TupleSelector
from vortex.handler.TupleDataObservableHandler import TuplesProviderABC

from peek_plugin_tutorial._private.storage.StringIntTuple import StringIntTuple

class StringIntTupleProvider(TuplesProviderABC):
    def __init__(self, ormSessionCreator):
        self._ormSessionCreator = ormSessionCreator

    @deferToThreadWrap
    def makeVortexMsg(self, filt: dict,
                     tupleSelector: TupleSelector) -> Union[Deferred, bytes]:
        # Potential filters can be placed here.
        # val1 = tupleSelector.selector["val1"]

        session = self._ormSessionCreator()
```

(continues on next page)

(continued from previous page)

```

try:
    tasks = (session.query(StringIntTuple)
              # Potentially filter the results
              # .filter(StringIntTuple.vall == vall)
              .all()
            )

    # Create the vortex message
    return Payload(filt, tuples=tasks).makePayloadEnvelope().toVortexMsg()

finally:
    session.close()

```

Edit File TupleDataObservable.py

Edit the TupleDataObservable.py python module, and register the new StringIntTupleProvider tuple provider.

Edit the file peek_plugin_tutorial/_private/server/TupleDataObservable.py:

1. Add the following imports:

```

from .tuple_providers.StringIntTupleProvider import StringIntTupleProvider
from peek_plugin_tutorial._private.storage.StringIntTuple import StringIntTuple

```

2. Find the line # Register TupleProviders here and add this line after it:

```

tupleObservable.addTupleProvider(StringIntTuple.tupleName(),
                                StringIntTupleProvider(ormSessionCreator))

```

Admin Update Notify

This section notifies the observable when an admin updates a StringIntTuple via the Admin service/UI.

This setup of the admin editing data, and having it change on Field/Office devices won't be the only way the observable is notified, however, it is a good setup for admin configurable items in dropdown lists, etc.

Edit File StringIntTableHandler.py

Edit the StringIntTableHandler.py file to accept the tupleObservable argument and notifyDeviceInfo the observable when an update occurs.

Edit the file peek_plugin_tutorial/_private/server/admin_backend/StringIntTableHandler.py

Add the import:

```

from vortex.TupleSelector import TupleSelector
from vortex.handler.TupleDataObservableHandler import TupleDataObservableHandler
from vortex.sqla_orm.OormCrudHandler import OormCrudHandlerExtension

```

Insert the following class, after the class definition of class `__CrudHandler`

```
class __ExtUpdateObservable(OrmCrudHandlerExtension):
    """ Update Observable ORM Crud Extension

    This extension is called after events that will alter data,
    it then notifies the observer.

    """
    def __init__(self, tupleDataObserver: TupleDataObservableHandler):
        self._tupleDataObserver = tupleDataObserver

    def _tellObserver(self, tuple_, tuples, session, payloadFilt):
        selector = {}
        # Copy any filter values into the selector
        # selector["lookupName"] = payloadFilt["lookupName"]
        tupleSelector = TupleSelector(StringIntTuple.tupleName(),
                                     selector)

        self._tupleDataObserver.notifyOfTupleUpdate(tupleSelector)
        return True

    afterUpdateCommit = _tellObserver
    afterDeleteCommit = _tellObserver
```

Update the instance of handler class

FROM

```
def makeStringIntTableHandler(dbSessionCreator):
```

TO

```
def makeStringIntTableHandler(tupleObservable, dbSessionCreator):
```

In the `makeStringIntTableHandler` method, insert this line just before the `return` handler

```
handler.addExtension(StringIntTuple, __ExtUpdateObservable(tupleObservable))
```

Edit File `admin_backend/__init__.py`

Edit `admin_backend/__init__.py` to take the observable parameter and pass it to the tuple provider handlers.

Edit file `peek_plugin_tutorial/_private/server/admin_backend/__init__.py`

Add the import:

```
from vortex.handler.TupleDataObservableHandler import TupleDataObservableHandler
```

Add the function call argument:

FROM

```
def makeAdminBackendHandlers(dbSessionCreator):
```

TO


```
def makeAdminBackendHandlers(tupleObservable: TupleDataObservableHandler,  
                             dbSessionCreator):
```

Pass the argument to the `makeStringIntTableHandler(...)` method:

FROM

```
yield makeStringIntTableHandler(dbSessionCreator)
```

TO

```
yield makeStringIntTableHandler(tupleObservable, dbSessionCreator)
```

Edit File `ServerEntryHook.py`

We need to update `ServerEntryHook.py`, to pass the new observable

Edit the file `peek_plugin_tutorial/_private/server/ServerEntryHook.py`, Add `tupleObservable` to the list of arguments passed to the `makeAdminBackendHandlers()` method:

FROM:

```
self._loadedObjects.extend(makeAdminBackendHandlers(self.dbSessionCreator))
```

TO:

```
self._loadedObjects.extend(  
    makeAdminBackendHandlers(tupleObservable, self.dbSessionCreator))
```

The tuple data observable will now notify `DeviceInfo` its observers when an admin updates the `StringInt` data.

Add Field View

Finally, lets add a new component to the Field screen.

Add Directory `string-int`

The `string-int` directory will contain the Angular component and views for our `stringInt` page.

Create the directory `peek_plugin_tutorial/_private/field-app/string-int` with the command:

```
mkdir peek_plugin_tutorial/_private/field-app/string-int
```

Add File `string-int.component.mweb.html`

The `string-int.component.mweb.html` file is the web app HTML **view** for the Angular component `string-int.component.ts`.

This is standard HTML with Angular directives.

Create the file `peek_plugin_tutorial/_private/field-app/string-int/string-int.component.mweb.html` and populate it with the following contents.

```
<div class="container">
  <Button class="btn btn-default" (click)="mainClicked()">Back to Main</Button>

  <table class="table table-striped">
    <thead>
      <tr>
        <th>String</th>
        <th>Int</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let item of stringInts">
        <td>{{item.string1}}</td>
        <td>{{item.int1}}</td>
      </tr>
    </tbody>
  </table>
</div>
```

Add File `string-int.component.ts`

The `string-int.component.ts` is the Angular Component that will be another route within the Tutorial plugin.

Create the file `peek_plugin_tutorial/_private/field-app/string-int/string-int.component.ts` and populate it with the following contents.

```
import {Component} from "@angular/core";
import {Router} from "@angular/router";
import {StringIntTuple, tutorialBaseUrl} from "@peek/peek_plugin_tutorial/_private";
import { NgLifecycleEvents } from "@synerty/peek-plugin-base-js"
import {
  TupleDataObserverService,
  TupleSelector
} from "@synerty/vortexjs";

@Component({
  selector: 'plugin-tutorial-string-int',
  templateUrl: 'string-int.component.mweb.html',
  moduleId: module.id
})
export class StringIntComponent extends NgLifecycleEvents {

  stringInts: Array<StringIntTuple> = [];
```

(continues on next page)

(continued from previous page)

```

constructor(private tupleDataObserver: TupleDataObserverService,
             private router: Router) {
    super();

    // Create the TupleSelector to tell the observable what data we want
    let selector = {};
    // Add any filters of the data here
    // selector["lookupName"] = "brownCowList";
    let tupleSelector = new TupleSelector(StringIntTuple.tupleName, selector);

    // Setup a subscription for the data
    let sup = tupleDataObserver.subscribeToTupleSelector(tupleSelector)
        .subscribe((tuples: StringIntTuple[]) => {
            // We've got new data, assign it to our class variable
            this.stringInts = tuples;
        });

    // unsubscribe when this component is destroyed
    // This is a feature of NgLifecycleEvents
    this.onDestroyEvent.subscribe(() => sup.unsubscribe());
}

mainClicked() {
    this.router.navigate([tutorialBaseUrl]);
}
}

```

Edit File `tutorial.module.ts`

Edit the `tutorial.module.ts`, to include the new component and add the route to it.

Edit `peek_plugin_tutorial/_private/field-app/tutorial.module.ts`:

1. Add the `StringIntComponent` import with the imports at the top of the file:

```
import {StringIntComponent} from "../string-int/string-int.component";
```

2. Insert the following as the first item in array `pluginRoutes`:

```
{
  path: 'stringint',
  component: StringIntComponent
},
```

3. Add the `StringIntComponent` to the declarations in the `@NgModule` decorator:

```
declarations: [...,
  StringIntComponent
], ...
```

4. Add the following to the Routes section:

```
{
  path: 'stringint',
  component: StringIntComponent
}
```

so it looks like below:

```
export const pluginRoutes: Routes = [
  ...
  {
    path: 'stringint',
    component: StringIntComponent
  }
  ...
]
```

At this point field is all setup, we just need to add some navigation buttons.

Edit File `tutorial.component.mweb.html`

Edit the web HTML view file, `tutorial.component.mweb.html` and insert a button that will change Angular Routes to our new component.

Edit file `peek_plugin_tutorial/_private/field-app/tutorial.component.mweb.html`, Insert the following just before the last closing `</div>` tag:

```
<Button class="btn btn-default"
  [routerLink]="['/peek_plugin_tutorial/stringint']">My Jobs >
</Button>
```

Testing

1. Open the field web app
2. Tap the Tutorial app icon
3. tap the “String Ints” button
4. Expect to see the string ints data.
5. Update the data from the Admin service UI
6. The data on the field app will immediately change.

Offline Observable

The Synerty VortexJS library has an `TupleDataOfflineObserverService`, once offline storage has been setup, (here [Add Offline Storage](#)), the offline observable is a dropin replacement.

When using the offline observable, it will:

1. Queue a request to observe the data, sending it to the field service

2. Query the SQL db in the browser/mobile device, and return the data for the observer. This provides instant data for the user.

When new data is sent to the the observer (Field/Office service) from the observable (Field service), the offline observer does two things:

1. Notifies the subscribers like normal
2. Stores the data back into the offline db, in the browser / app.

Edit File `string-int.component.ts`

`TupleDataOfflineObserverService` is a drop-in replacement for `TupleDataObserverService`.

Switching to use the offline observer requires two edits to `string-int.component.ts`.

Edit file `peek_plugin_tutorial/_private/field-app/string-int/string-int.component.ts`.

Add the import for the `TupleDataOfflineObserverService`:

```
import {TupleDataOfflineObserverService} from "@synerty/vortexjs";
```

Change the type of the `tupleDataObserver` parameter in the component constructor, EG,

From

```
constructor(private tupleDataObserver: TupleDataObserverService, ...) {
```

To

```
constructor(private tupleDataObserver: TupleDataOfflineObserverService, ...) {
```

That's it. Now the String Int data will load on the device, even when the Vortex between the device and the field service is offline.

Add More Observables

This was a long tutorial, but the good news is that you don't have to repeat all this every time. Here are the steps you need to repeat to observe more data, altering them to suit of course.

Create the Python tuples, either *[Adding a StringInt Table](#)* or *[Add File TutorialTuple.py](#)*

Add the TypeScript tuples, *[Edit File plugin_package.json](#)*.

Add a Logic service tuple provider, *[Add Tuple Provider](#)*

Then, add the Field, Office or Admin side, add the views and Angular component, *[Add Field View](#)*.

3.1.16 Add Actions

Outline

In this document we setup the VortexJS Tuple Actions.

Since the Vortex serialisable base class is called a `Tuple`, Actions are referred to as “Action Tuples”, and name `DoSomethingActionTuple`.

A Tuple Action represents an action the user has taken, this can be:

- Clicking a button (`TupleGenericAction`)
- Updating data (`TupleUpdateAction`)
- Some other action (extend `TupleActionABC`)

The Action design is ideal for apps where there are many users observing data more than altering it or performing actions against it.

Typically, users can only perform so many updates per a minute. `TupleActions` takes the approach of having many small, discrete “Actions” that can be sent back to the server as they are performed.

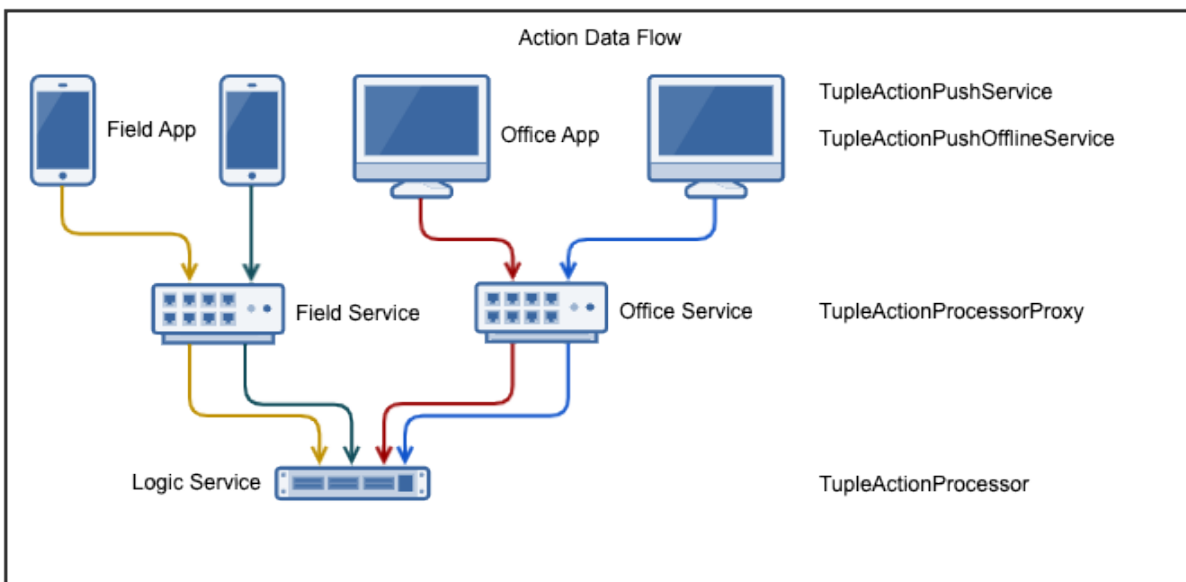
The Observable then ensures that all users watching the data are updated immediately, Keeping all users working with the latest data as `TupleActions` processed.

This helps avoid issues, such as one user’s update overwriting another user’s update. These issues you will get if you’re using the VortexJS `TupleLoader` for many users.

There are two Angular services that provide support for pushing Tuple Actions to the Field service.

1. `TupleActionPushService`, for online only actions.
2. `TupleActionPushOfflineService`, for actions that will be stored locally and delivered when the device is next online.

Both these services have the same functional interface, `pushAction()`.



On the Logic service, the `TupleActionProcessorProxy` class receives all the `TupleActions`, delegates processing to a `TupleActionProcessorDelegateABC` class. A delegate can be registered to handle just one type of

action, and/or a default delegate can be registered to catch all.

Like the Observable, there is a `TupleActionProcessorProxy` needed in the field service that passes actions onto the Logic service for processing.

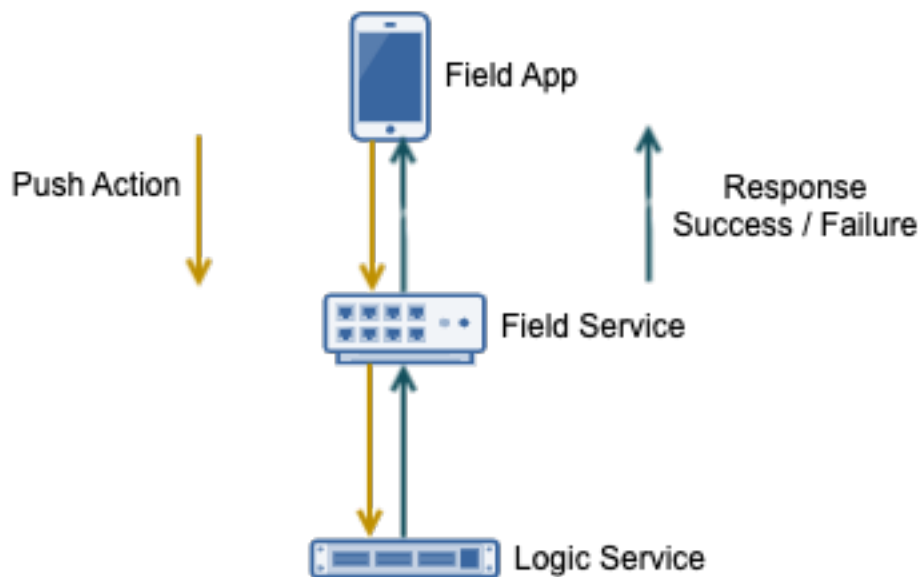
Unlike the Observable, the `TupleAction` Field or Office proxy passes every action onto the Logic service, waits for a response from the Logic service then sends that back to the Field or Office device.

Actions require responses. Callers of the `TupleActionPushService` will receive a promise which resolve regardless of if the push timed out or failed.

In the case of `TupleActionPushOfflineService`, a promise is returned and resolved on success of the commit to the database in the Field/Office device.

The `TupleActionPushOfflineService` will continually retry until it receives either a success or failure response from the Field service.

Note: The Field/Office devices don't and can't talk directly to the Logic service.



Advantages

1. Reduces the risk of one update overwriting another.
2. Atomic changes can more easily be buffered when the device is offline.
3. Smaller, more immediate results for updates.

Disadvantages

1. This could lead to higher resource usage and less efficient commits.

Objective

In this document, our plugin will provide the following actions to the user:

1. Increase or decrease an Int
2. Toggle capitals of a string

The action will be processed by the Logic Service which will update the table created in *Adding a StringInt Table*.

This is the order:

1. Add the Action scaffolding for the project.
2. Add the Logic side Action Processor
3. Alter the Observable tutorial UI to incorporate buttons and send the actions.

Add Python Tuples

Add File `StringCapToggleActionTuple.py`

The `StringCapToggleActionTuple.py` defines a python action tuple.

Create the file `peek_plugin_tutorial/_private/tuples/StringCapToggleActionTuple.py` and populate it with the following contents.

```
from vortex.Tuple import addTupleType, TupleField
from vortex.TupleAction import TupleActionABC

from peek_plugin_tutorial._private.PluginNames import tutorialTuplePrefix

@addTupleType
class StringCapToggleActionTuple(TupleActionABC):
    __tupleType__ = tutorialTuplePrefix + "StringCapToggleActionTuple"

    stringIntId = TupleField()
```

Add File `AddIntValueActionTuple.py`

The `AddIntValueActionTuple.py` defines a python action tuple.

Create the file `peek_plugin_tutorial/_private/tuples/AddIntValueActionTuple.py` and populate it with the following contents.

```
from vortex.Tuple import addTupleType, TupleField
from vortex.TupleAction import TupleActionABC

from peek_plugin_tutorial._private.PluginNames import tutorialTuplePrefix

@addTupleType
class AddIntValueActionTuple(TupleActionABC):
```

(continues on next page)

(continued from previous page)

```
__tupleType__ = tutorialTuplePrefix + "AddIntValueActionTuple"

stringIntId = TupleField()
offset = TupleField()
```

Add TypeScript Tuples

Add StringCapToggleActionTuple.ts

The StringCapToggleActionTuple.ts file defines a TypeScript class for our StringCapToggleActionTuple Tuple Action.

Create file peek_plugin_tutorial/plugin-module/_private/tuples/StringCapToggleActionTuple.ts, with contents

```
import {addTupleType, Tuple, TupleActionABC} from "@synerty/vortexjs";
import {tutorialTuplePrefix} from "../PluginNames";

@addTupleType
export class StringCapToggleActionTuple extends TupleActionABC {
    static readonly tupleName = tutorialTuplePrefix + "StringCapToggleActionTuple";

    stringIntId: number;

    constructor() {
        super(StringCapToggleActionTuple.tupleName)
    }
}
```

Add AddIntValueActionTuple.ts

The AddIntValueActionTuple.ts file defines a TypeScript class for our AddIntValueActionTuple Tuple Action.

Create file peek_plugin_tutorial/plugin-module/_private/tuples/AddIntValueActionTuple.ts, with contents

```
import {addTupleType, Tuple, TupleActionABC} from "@synerty/vortexjs";
import {tutorialTuplePrefix} from "../PluginNames";

@addTupleType
export class AddIntValueActionTuple extends TupleActionABC {
    public static readonly tupleName = tutorialTuplePrefix + "AddIntValueActionTuple";

    stringIntId: number;
    offset: number;

    constructor() {
        super(AddIntValueActionTuple.tupleName)
    }
}
```

Edit File `_private/index.ts`

The `_private/index.ts` file will re-export the Tuples in a more standard way. Developers won't need to know the exact path of the file.

Edit file `peek_plugin_tutorial/plugin-module/_private/index.ts`, Append the lines:

```
export {StringCapToggleActionTuple} from "../tuples/StringCapToggleActionTuple";
export {AddIntValueActionTuple} from "../tuples/AddIntValueActionTuple";
```

Logic Service Setup

Add Package controller

The `controller` python package will contain the classes that provide logic to the plugin, like a brain controlling limbs.

Note: Though the tutorial creates “controllers”, the plugin developer can decide how ever they want to structure this.

Create the `peek_plugin_tutorial/_private/logic/controller` package, with the commands

```
mkdir peek_plugin_tutorial/_private/logic/controller
touch peek_plugin_tutorial/_private/logic/controller/__init__.py
```

Add File `MainController.py`

The `MainController.py` will glue everything together. For large plugins there will be multiple sub controllers.

In this example we have everything in `MainController`.

Create the file `peek_plugin_tutorial/_private/logic/controller/MainController.py` and populate it with the following contents.

```
import logging

from twisted.internet.defer import Deferred
from vortex.DeferUtil import deferToThreadWrapWithLogger

from vortex.TupleSelector import TupleSelector
from vortex.TupleAction import TupleActionABC
from vortex.handler.TupleActionProcessor import TupleActionProcessorDelegateABC
from vortex.handler.TupleDataObservableHandler import TupleDataObservableHandler

from peek_plugin_tutorial._private.storage.StringIntTuple import StringIntTuple
from peek_plugin_tutorial._private.tuples.StringCapToggleActionTuple import _
    ↳StringCapToggleActionTuple
from peek_plugin_tutorial._private.tuples.AddIntValueActionTuple import _
    ↳AddIntValueActionTuple
```

(continues on next page)

(continued from previous page)

```

logger = logging.getLogger(__name__)

class MainController(TupleActionProcessorDelegateABC):
    def __init__(self, dbSessionCreator, tupleObservable: TupleDataObservableHandler):
        self._dbSessionCreator = dbSessionCreator
        self._tupleObservable = tupleObservable

    def shutdown(self):
        pass

    def processTupleAction(self, tupleAction: TupleActionABC) -> Deferred:

        if isinstance(tupleAction, AddIntValueActionTuple):
            return self._processAddIntValue(tupleAction)

        if isinstance(tupleAction, StringCapToggleActionTuple):
            return self._processCapToggleString(tupleAction)

        raise NotImplementedError(tupleAction.tupleName())

@deferToThreadWrapWithLogger(logger)
def _processCapToggleString(self, action: StringCapToggleActionTuple):
    try:
        # Perform update using SQLAlchemy
        session = self._dbSessionCreator()
        row = (session.query(StringIntTuple)
                .filter(StringIntTuple.id == action.stringIntId)
                .one())

        # Exit early if the string is empty
        if not row.string1:
            logger.debug("string1 for StringIntTuple.id=%s is empty")
            return

        if row.string1[0].isupper():
            row.string1 = row.string1.lower()
            logger.debug("Toggled to lower")
        else:
            row.string1 = row.string1.upper()
            logger.debug("Toggled to upper")

        session.commit()

        # Notify the observer of the update
        # This tuple selector must exactly match what the UI observes
        tupleSelector = TupleSelector(StringIntTuple.tupleName(), {})
        self._tupleObservable.notifyOfTupleUpdate(tupleSelector)

    finally:
        # Always close the session after we create it
        session.close()

@deferToThreadWrapWithLogger(logger)
def _processAddIntValue(self, action: AddIntValueActionTuple):
    try:

```

(continues on next page)

(continued from previous page)

```
# Perform update using SQLAlchemy
session = self._dbSessionCreator()
row = (session.query(StringIntTuple)
        .filter(StringIntTuple.id == action.stringIntId)
        .one())
row.int1 += action.offset
session.commit()

logger.debug("Int changed by %u", action.offset)

# Notify the observer of the update
# This tuple selector must exactly match what the UI observes
tupleSelector = TupleSelector(StringIntTuple.tupleName(), {})
self._tupleObservable.notifyOfTupleUpdate(tupleSelector)

finally:
    # Always close the session after we create it
    session.close()
```

Add File TupleActionProcessor.py

The class in file TupleActionProcessor.py, accepts all tuple actions for this plugin and calls the relevant TupleActionProcessorDelegateABC.

Create the file peek_plugin_tutorial/_private/logic/TupleActionProcessor.py and populate it with the following contents.

```
from vortex.handler.TupleActionProcessor import TupleActionProcessor

from peek_plugin_tutorial._private.PluginNames import tutorialFilt
from peek_plugin_tutorial._private.PluginNames import tutorialActionProcessorName
from .controller.MainController import MainController

def makeTupleActionProcessorHandler(mainController: MainController):
    processor = TupleActionProcessor(
        tupleActionProcessorName=tutorialActionProcessorName,
        additionalFilt=tutorialFilt,
        defaultDelegate=mainController)
    return processor
```

Edit File LogicEntryHook.py

We need to update LogicEntryHook.py, it will initialise the MainController and TupleActionProcessor objects.

Edit the file peek_plugin_tutorial/_private/logic/LogicEntryHook.py:

1. Add these imports at the top of the file with the other imports:

```
from .TupleActionProcessor import makeTupleActionProcessorHandler
from .controller.MainController import MainController
```

2. Add these line just before `logger.debug("started")` in the `start()` method:

```
mainController = MainController(
    dbSessionCreator=self.dbSessionCreator,
    tupleObservable=tupleObservable)

self._loadedObjects.append(mainController)
self._loadedObjects.append(makeTupleActionProcessorHandler(mainController))
```

The Action Processor for the Logic Service service is setup now.

Field Service Setup

Add File DeviceTupleProcessorActionProxy.py

The `DeviceTupleProcessorActionProxy.py` creates the Tuple Action Processor Proxy. This class is responsible for proxying action tuple data between the devices and the Logic Service.

Create the file `peek_plugin_tutorial/_private/field/DeviceTupleProcessorActionProxy.py` and populate it with the following contents.

```
from peek_plugin_base.PeekVortexUtil import peekServerName
from peek_plugin_tutorial._private.PluginNames import tutorialFilt
from peek_plugin_tutorial._private.PluginNames import tutorialActionProcessorName
from vortex.handler.TupleActionProcessorProxy import TupleActionProcessorProxy

def makeTupleActionProcessorProxy():
    return TupleActionProcessorProxy(
        tupleActionProcessorName=tutorialActionProcessorName,
        proxyToVortexName=peekServerName,
        additionalFilt=tutorialFilt)
```

Edit File FieldEntryHook.py

We need to update `FieldEntryHook.py`, it will initialise the tuple action proxy object when the Plugin is started.

Edit the file `peek_plugin_tutorial/_private/field/FieldEntryHook.py`:

1. Add this import at the top of the file with the other imports:

```
from .DeviceTupleProcessorActionProxy import makeTupleActionProcessorProxy
```

2. Add this line after the docstring in the `start()` method:

```
self._loadedObjects.append(makeTupleActionProcessorProxy())
```

Office Service Setup

Add File DeviceTupleProcessorActionProxy.py

The DeviceTupleProcessorActionProxy.py creates the Tuple Action Processor Proxy. This class is responsible for proxying action tuple data between the devices and the Logic Service.

Create the file peek_plugin_tutorial/_private/office/DeviceTupleProcessorActionProxy.py and populate it with the following contents.

```
from peek_plugin_base.PeekVortexUtil import peekServerName
from peek_plugin_tutorial._private.PluginNames import tutorialFilt
from peek_plugin_tutorial._private.PluginNames import tutorialActionProcessorName
from vortex.handler.TupleActionProcessorProxy import TupleActionProcessorProxy

def makeTupleActionProcessorProxy():
    return TupleActionProcessorProxy(
        tupleActionProcessorName=tutorialActionProcessorName,
        proxyToVortexName=peekServerName,
        additionalFilt=tutorialFilt)
```

Edit File OfficeEntryHook.py

We need to update OfficeEntryHook.py, it will initialise the tuple action proxy object when the Plugin is started.

Edit the file peek_plugin_tutorial/_private/office/OfficeEntryHook.py:

1. Add this import at the top of the file with the other imports:

```
from .DeviceTupleProcessorActionProxy import makeTupleActionProcessorProxy
```

2. Add this line after the docstring in the start() method:

```
self._loadedObjects.append(makeTupleActionProcessorProxy())
```

Field App Setup

Now we need to edit the Angular module in the field-app and add the providers:

Edit File tutorial.module.ts

Edit the tutorial.module.ts Angular module for the tutorial plugin to add the provider entry for the TupleAction service.

Edit the file peek_plugin_tutorial/_private/field-app/tutorial.module.ts:

1. Add the following imports:

```
// Import the required classes from VortexJS
import {
  TupleActionPushNameService,
  TupleActionPushOfflineService,
  TupleActionPushService
} from "@synerty/vortexjs";

// Import the names we need for the
import {
  tutorialActionProcessorName
} from "@peek/peek_plugin_tutorial/_private";
```

2. After the imports, add this function

```
export function tupleActionPushNameServiceFactory() {
  return new TupleActionPushNameService(
    tutorialActionProcessorName, tutorialFilt);
}
```

3. Finally, add this snippet to the providers array in the @NgModule decorator

```
TupleActionPushOfflineService, TupleActionPushService, {
  provide: TupleActionPushNameService,
  useFactory: tupleActionPushNameServiceFactory
},
```

It should look similar to the following:

```
...

import {
  TupleActionPushNameService,
  TupleActionPushOfflineService,
  TupleActionPushService
} from "@synerty/vortexjs";

import {
  tutorialActionProcessorName
} from "@peek/peek_plugin_tutorial/_private";

...

export function tupleActionPushNameServiceFactory() {
  return new TupleActionPushNameService(
    tutorialActionProcessorName, tutorialFilt);
}

@NgModule({
  ...
  providers: [
    ...
    TupleActionPushOfflineService, TupleActionPushService, {
      provide: TupleActionPushNameService,
      useFactory: tupleActionPushNameServiceFactory
    },
    ...
```

(continues on next page)

(continued from previous page)

```
    ]
  })
  export class TutorialModule {
  }
```

At this point, all of the Tuple Action setup is done. It's much easier to work with the tuple action code from here on.

Add Mobile View

Finally, lets add a new component to the mobile screen.

Edit File `string-int.component.ts`

Edit the file, `string-int.component.ts` to connect the tuple action to the frontend.

edit the file `peek_plugin_tutorial/_private/field-app/string-int/string-int.component.ts`

1. Add the following imports:

```
import {TupleActionPushService} from "@synerty/vortexjs";

import {
  AddIntValueActionTuple,
  StringCapToggleActionTuple
} from "@peek/peek_plugin_tutorial/_private";
```

1. Add private `actionService: TupleActionPushService` to the constructor argument:

```
constructor(private actionService: TupleActionPushService,
  ...) {
```

1. Finally, add the methods to the `StringIntComponent` class after the constructor:

```
toggleUpperClicked(item) {
  let action = new StringCapToggleActionTuple();
  action.stringIntId = item.id;
  this.actionService.pushAction(action)
    .then(() => {
      alert('success');
    })
    .catch((err) => {
      alert(err);
    });
}

incrementClicked(item) {
  let action = new AddIntValueActionTuple();
  action.stringIntId = item.id;
```

(continues on next page)

(continued from previous page)

```

    action.offset = 1;
    this.actionService.pushAction(action)
      .then(() => {
        alert('success');

      })
      .catch((err) => {
        alert(err);
      });
  }

  decrementClicked(item) {
    let action = new AddIntValueActionTuple();
    action.stringIntId = item.id;
    action.offset = -1;
    this.actionService.pushAction(action)
      .then(() => {
        alert('success');

      })
      .catch((err) => {
        alert(err);
      });
  }
}

```

It should look similar to the following:

```

...

import {
  AddIntValueActionTuple,
  StringCapToggleActionTuple
} from "@peek/peek_plugin_tutorial/_private";

...

constructor(private actionService: TupleActionPushService,
  ...) {

  ...

  incrementClicked(item) {
    let action = new AddIntValueActionTuple();
    action.stringIntId = item.id;
    action.offset = 1;
    this.actionService.pushAction(action)
      .then(() => {
        alert('success');

      })
      .catch((err) => {
        alert(err);
      });
  }

  decrementClicked(item) {

```

(continues on next page)

(continued from previous page)

```

    let action = new AddIntValueActionTuple();
    action.stringIntId = item.id;
    action.offset = -1;
    this.actionService.pushAction(action)
      .then(() => {
        alert('success');

      })
      .catch((err) => {
        alert(err);
      });
  }

  mainClicked() {
    this.router.navigate([tutorialBaseUrl]);
  }
}

```

Edit File `string-int.component.mweb.html`

Edit the web HTML view file, `string-int.component.mweb.html` and insert buttons that will change initiate the created tuple actions.

Edit the file `peek_plugin_tutorial/_private/field-app/string-int/string-int.component.mweb.html` and populate it with the following contents:

```

<div class="container">
  <Button class="btn btn-default" (click)="mainClicked()">Back to Main</Button>

  <table class="table table-striped">
    <thead>
      <tr>
        <th>String</th>
        <th>Int</th>
        <th></th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let item of stringInts">
        <td>{{item.string1}}</td>
        <td>{{item.int1}}</td>
        <td>
          <Button class="btn btn-default" (click)="toggleUpperClicked(item)">
            Toggle Caps
          </Button>
          <Button class="btn btn-default" (click)="incrementClicked(item)">
            Increment Int
          </Button>
          <Button class="btn btn-default" (click)="decrementClicked(item)">
            Decrement Int
          </Button>
        </td>
      </tr>
    </tbody>
  </table>

```

(continues on next page)

(continued from previous page)

```

        </tr>
      </tbody>
    </table>
  </div>

```

Testing

1. Open Field web app
2. Tap the Tutorial app icon
3. Tap the “String Ints” button
4. Expect to see the string ints data
5. Select the “Toggle Caps” button
6. If successful an alert will appear stating “success”. If you receive an error, go back through the “Add Actions” instructions. Restart the logic service and retry step five
7. You will see the data update instantly
8. Return to step five for buttons “Increment Int” and “Decrement Int”

Offline Observable

The Synerty VortexJS library has an `TupleDataOfflineObserverService`, once offline storage has been setup, (here [Add Offline Storage](#)), the offline observable is a drop in replacement.

When using the offline observable, it will:

1. Queue a request to observe the data, sending it to the field service
2. Query the SQL db in the browser/field device, and return the data for the observer. This provides instant data for the user.

When new data is sent to the the observer (Field/Office service) from the observable (Field Service), the offline observer does two things:

1. Notifies the subscribers like normal
2. Stores the data back into the offline db, in the browser / app.

Edit File `string-int.component.ts`

`TupleDataOfflineObserverService` is a drop-in replacement for `TupleDataObserverService`.

Switching to use the offline observer requires two edits to `string-int.component.ts`.

Edit file `peek_plugin_tutorial/_private/field-app/string-int/string-int.component.ts`.

Add the import for the `TupleDataOfflineObserverService`:

```
import TupleDataOfflineObserverService from "@synerty/vortexjs";
```

Change the type of the `tupleDataObserver` parameter in the component constructor, EG,

From

```
constructor(private tupleDataObserver: TupleDataObserverService, ...) {
```

To

```
constructor(private tupleDataObserver: TupleDataOfflineObserverService, ...) {
```

That's it. Now the String Int data will load on the device, even when the Vortex between the device and the field service is offline.

3.1.17 Add Vortex RPC

Outline

Peek has distributed services, and one plugin is usually run on more than one of these services. This can make it incredibly complicated for code within the plugin that runs on the agent service, to talk to the logic service for example.

In this document, we go through using the Vortex RPC to simplify communications between the logic and agent service.

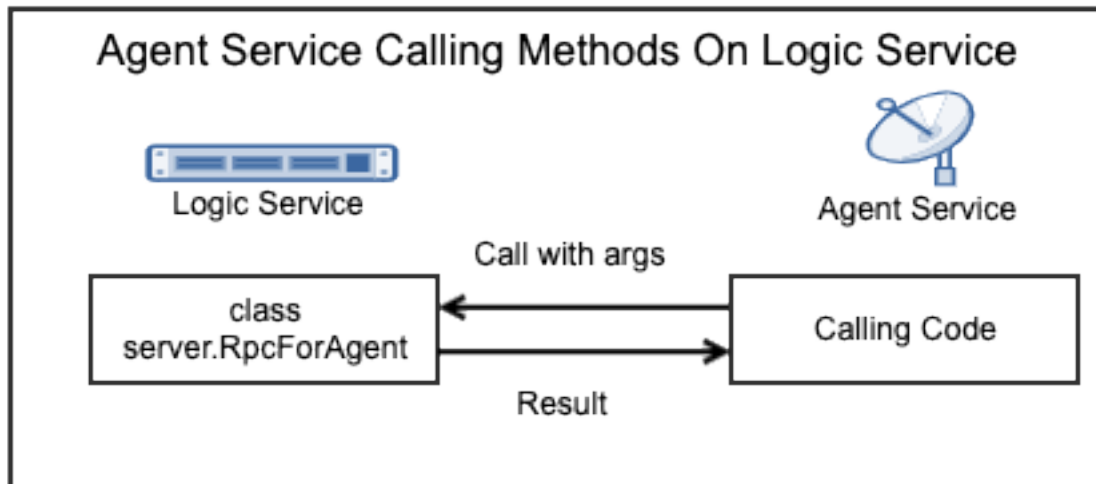
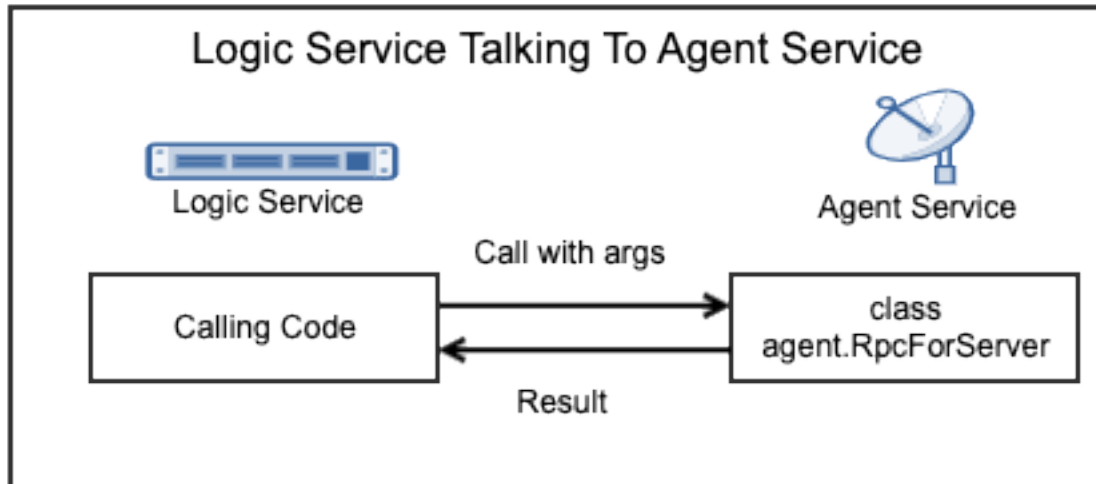
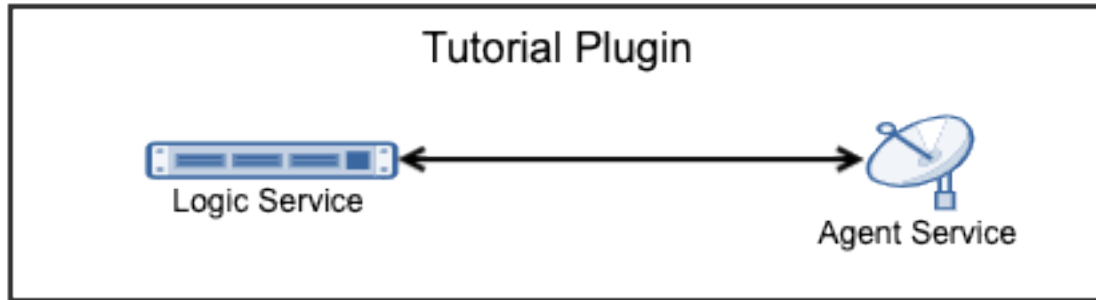
What is RPC

RPC stands for Remote Procedure Call, essentially it allows you to call methods/functions/procedure over the network to another process.

In this example, the two processes are the Agent service and the Logic service. These are completely separate processes, so you can't just call a method defined in the logic service from the agent service.

Vortex RPC provides wrappers that make it easy to define procedures in one service, and call them from another.

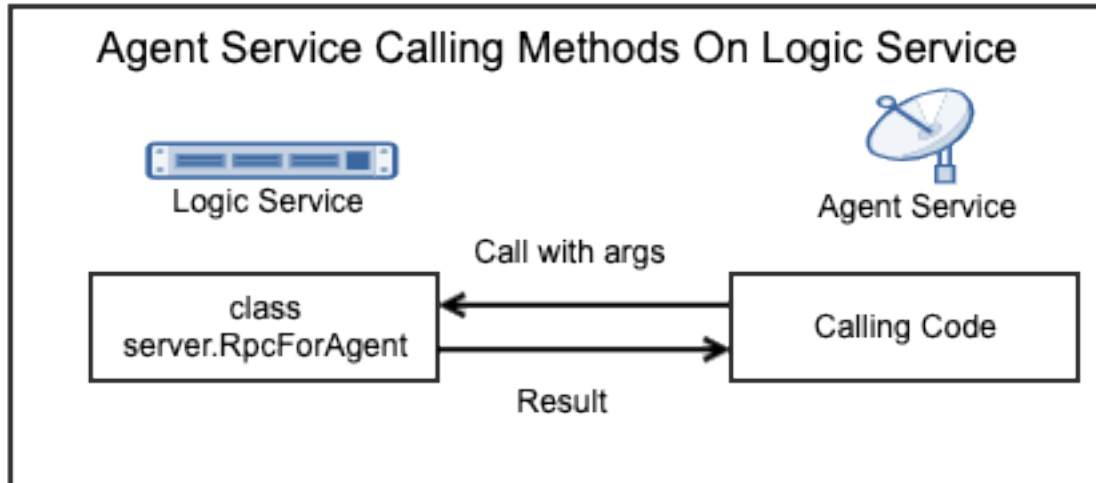
Note: Vortex RPC calls return `twisted.internet.defer.Deferred`, regardless of what the actual method returns.



Logic Service RPC Setup

In this section we setup the files required to define an RPC on the logic service that will only accept calls from the agent.

The RPC example could be much simpler, the intention is to show more of a good design verses the bare minimum RPC example.



Add Package `agent_handlers`

The `agent_handlers` python package will contain the classes that generate tuple data to send via the observable.

Create the `peek_plugin_tutorial/_private/logic/agent_handlers` package, with the commands

```
mkdir peek_plugin_tutorial/_private/logic/agent_handlers
touch peek_plugin_tutorial/_private/logic/agent_handlers/__init__.py
```

Add File `RpcForAgent.py`

File `RpcForAgent.py` defines the methods the agent will call via RPC.

In this example we have just one file, however it will be good practice to have multiple files if the require RPC methods grow too large.

Create the file `peek_plugin_tutorial/_private/logic/agent_handlers/RpcForAgent.py` and populate it with the following contents.

```
import logging

from peek_plugin_base.PeekVortexUtil import peekServerName, peekAgentName
from peek_plugin_tutorial._private.PluginNames import tutorialFilt
from peek_plugin_tutorial._private.logic.controller.MainController import _
↳MainController
from peek_plugin_tutorial._private.storage.StringIntTuple import StringIntTuple
from vortex.rpc.RPC import vortexRPC

logger = logging.getLogger(__name__)

class RpcForAgent:
```

(continues on next page)

(continued from previous page)

```

def __init__(self, mainController: MainController, dbSessionCreator):
    self._mainController = mainController
    self._dbSessionCreator = dbSessionCreator

def makeHandlers(self):
    """ Make Handlers

    In this method we start all the RPC handlers
    start() returns an instance of itself so we can simply yield the result
    of the start method.

    """

    yield self.addInts.start(funcSelf=self)
    yield self.updateStatus.start(funcSelf=self)
    yield self.addStringInt.start(funcSelf=self)
    logger.debug("RPCs started")

# -----
@vortexRPC(peekServerName,
           acceptOnlyFromVortex=peekAgentName, additionalFilt=tutorialFilt)
def addInts(self, val1, kwval1=9):
    """ Add Ints

    This is the simplest RPC example possible

    """
    return val1 + kwval1

# -----
@vortexRPC(peekServerName,
           acceptOnlyFromVortex=peekAgentName, additionalFilt=tutorialFilt)
def updateStatus(self, updateStr: str):
    """ Update Status

    The agent may be running something and send updates on occasion,
    tell these to the main controller, it can deal with them.

    """
    self._mainController.agentNotifiedOfUpdate(updateStr)

# -----
@vortexRPC(peekServerName, acceptOnlyFromVortex=peekAgentName,
           additionalFilt=tutorialFilt, deferToThread=True)
def addStringInt(self, stringInt: StringIntTuple):
    """ Insert a stringInt

    In this example RPC method, The agent tells the logic service to insert data
    ↪ into
    the database.

    It's a better design get the main controller to do things like this.
    It will know what else needs updating after the insert (IE, The observable)

    Notice the :code:`deferToThread=True` argument in :code:`@vortexRPC`?
    Because this code is blocking code, not written for twisted, we need to
    defer it to a thread so it doesn't block twisted's main reactor.

```

(continues on next page)

(continued from previous page)

```
As it's no longer in the twisted thread, all the code in this method  
should be standard blocking code.
```

```
"""  
session = self._dbSessionCreator()  
try:  
    session.add(stringInt)  
  
except:  
    session.rollback()  
    raise  
  
finally:  
    session.close()
```

Edit File MainController.py

We need to update `MainController.py`, to add an example method that the `RpcForAgent` will call.

Edit the file `peek_plugin_tutorial/_private/logic/controller/MainController.py`:

1. Add this line to the bottom of the file, inside the class definition:

```
def agentNotifiedOfUpdate(self, updateStr):  
    logger.debug("Agent said : %s", updateStr)
```

Edit File LogicEntryHook.py

We need to update `LogicEntryHook.py`, to initialise the `RpcForAgent`.

Edit the file `peek_plugin_tutorial/_private/logic/LogicEntryHook.py`:

1. Add this import at the top of the file with the other imports:

```
from .agent_handlers.RpcForAgent import RpcForAgent
```

2. Add this line just before the `logger.debug("Started")` line at the end of the `start()` method:

```
# Initialise the RpcForAgent  
self._loadedObjects.extend(RpcForAgent(mainController, self.dbSessionCreator)  
                           .makeHandlers())
```

The sever side RPC is now setup.

Agent Calling Logic Service RPC

This section implements the code in the agent that will call the RPC methods that the logic service has defined.

Add File AgentToLogicRpcCallExample.py

File AgentToLogicRpcCallExample.py defines the methods the agent will call via RPC.

In this example we have just one file, however it will be good practice to have multiple files if the require RPC methods grow too large.

Create the file peek_plugin_tutorial/_private/agent/AgentToLogicRpcCallExample.py and populate it with the following contents.

```
import logging

from twisted.internet import reactor
from twisted.internet.defer import inlineCallbacks

from peek_plugin_tutorial._private.logic.agent_handlers.RpcForAgent import RpcForAgent
from peek_plugin_tutorial._private.storage.StringIntTuple import StringIntTuple

logger = logging.getLogger(__name__)

class AgentToLogicRpcCallExample:
    def start(self):
        # kickoff the example
        # Tell the reactor to start it in 5 seconds, we shouldn't do things like
        # this in the plugins start method.
        reactor.callLater(5, self.runWithInlineCallback)

        # Return self, to make it simpler for the AgentEntryHook
        return self

    @inlineCallbacks
    def runWithInlineCallback(self):
        """ Run With Inline Callbacks

        To understand what the :code:`@inlineCallbacks` decorator does, you can read
        more in the twisted documentation.

        This is the simplest way to go with asynchronous code.

        Yield here, will cause the flow of code to return to the twisted.reactor
        until the deferreds callback or errback is called.

        The errback will cause an exception, which we'd catch with a standard
        try/except block.

        """

        # The :code:`@vortexRPC` decorator wraps the :code:`RpcForAgent.updateStatus`
        # method with an instance of the :code:`_VortexRPC` class,
        # this class has a :code:`__call__` method implemented, that is what we're
        # calling here.
        #
        # So although it looks like we're trying to call a class method, that's not
        ↪ what's
        # happening.
```

(continues on next page)

(continued from previous page)

```

yield RpcForAgent.updateStatus("Agent RPC Example Started")

seedInt = 5
logger.debug("seedInt = %s", seedInt)

for _ in range(5):
    seedInt = yield RpcForAgent.addInts(seedInt, kwval1=7)
    logger.debug("seedInt = %s", seedInt)

# Move onto the run method.
# We don't use yield here, so :code:`runWithInlineCallback` will continue on_
↪and
# finish
self.run()
logger.debug("runWithInlineCallback finished")

def run(self):
    """ Run

    In this method, we call some RPCs and handle the deferreds.

    We won't be using @inlineCallbacks here. We will setup all the calls and
    callbacks, then the run method will return. The calls and callbacks will_
↪happen
    long after this method finishes.

    """

    stringInt = StringIntTuple(int1=50, string1="Created from Agent RPC")

    d = RpcForAgent.addStringInt(stringInt)

    # the deferred will call the lambda function,
    #   "_" will be the result of "addStringInt, which we ignore
    #   the lambda function calls RpcForAgent.updateStatus,
    #   which will return a deferred
    #
    # Returning a deferred from a callback is fine, it's just merilly processed
    d.addCallback(lambda _: RpcForAgent.updateStatus("Agent RPC Example Completed
↪"))

    # Unless you have a good reason, always return the last deferred.
    return d

def shutdown(self):
    pass

```

Edit File AgentEntryHook.py

We need to update AgentEntryHook.py, to initialise the AgentToLogicRpcCallExample.

Edit the file peek_plugin_tutorial/_private/agent/AgentEntryHook.py:

1. Add this import at the top of the file with the other imports:

```
from .AgentToLogicRpcCallExample import AgentToLogicRpcCallExample
```

2. Add this line just before the `logger.debug("Started")` line at the end of the `start()` method:

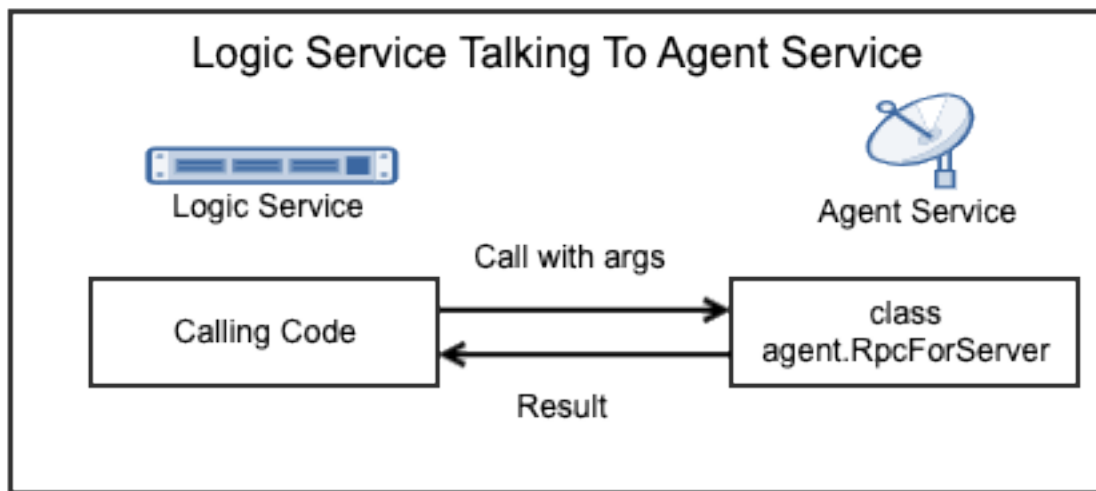
```
# Initialise and start the AgentToLogicRpcCallExample
self._loadedObjects.append(AgentToLogicRpcCallExample().start())
```

The agent will now call the logic service RPC methods.

Agent RPC Setup

In this section we setup the files required to define an RPC on the agent that the logic service will call.

Some example use cases would be: * Agent to query data from external DB * Agent to connect to remote server via SSH and pull back some data * Agent to push an update to a corporate system via HTTP



Add File `RpcForLogic.py`

File `RpcForLogic.py` defines the methods the logic service will call via RPC.

Create the file `peek_plugin_tutorial/_private/agent/RpcForLogic.py` and populate it with the following contents.

```
import logging

from peek_plugin_base.PeekVortexUtil import peekAgentName
from peek_plugin_tutorial._private.PluginNames import tutorialFilt
from vortex.rpc.RPC import vortexRPC

logger = logging.getLogger(__name__)
```

(continues on next page)

(continued from previous page)

```
class RpcForLogic:
    def __init__(self):
        pass

    def makeHandlers(self):
        """ Make Handlers

        In this method we start all the RPC handlers
        start() returns an instance of itself so we can simply yield the result
        of the start method.

        """

        yield self.subInts.start(funcSelf=self)
        logger.debug("LogicService RPCs started")

# -----
@vortexRPC(peekAgentName, additionalFilt=tutorialFilt)
def subInts(self, val1, kwval1=9):
    """ Add Ints

    This is the simplest RPC example possible.

    :param val1: A value to start with
    :param kwval1: The value to subtract
    :return: One value minus the other

    """
    return val1 - kwval1
```

Edit File AgentEntryHook.py

We need to update AgentEntryHook.py, to initialise the RpcForLogic.

Edit the file peek_plugin_tutorial/_private/agent/AgentEntryHook.py:

1. Add this import at the top of the file with the other imports:

```
from .RpcForLogic import RpcForLogic
```

2. Add this line just before the logger.debug("Started") line at the end of the start() method:

```
# Initialise and start the RPC for Logic Service
self._loadedObjects.extend(RpcForLogic().makeHandlers())
```

The sever side RPC is now setup.

Logic Service Calling Agent RPC

This section implements the code in the logic service that will call the RPC methods that the agent has defined.

Add File LogicToAgentRpcCallExample.py

File `LogicToAgentRpcCallExample.py` defines the methods the logic service will call via RPC.

Create the file `peek_plugin_tutorial/_private/logic/LogicToAgentRpcCallExample.py` and populate it with the following contents.

```
import logging

from twisted.internet import reactor
from twisted.internet.defer import inlineCallbacks

from peek_plugin_tutorial._private.agent.RpcForLogic import RpcForLogic

logger = logging.getLogger(__name__)

class LogicToAgentRpcCallExample:
    def start(self):
        # kickoff the example
        # Tell the reactor to start it in 20 seconds, we shouldn't do things like
        # this in the plugins start method.
        reactor.callLater(20, self.run)

        return self

    @inlineCallbacks
    def run(self):
        # Call the agents RPC method
        result = yield RpcForLogic.subInts(7, kwall=5)
        logger.debug("seedInt result = %s (Should be 2)", result)

    def shutdown(self):
        pass
```

Edit File LogicEntryHook.py

We need to update `LogicEntryHook.py`, to initialise the `LogicToAgentRpcCallExample`.

Edit the file `peek_plugin_tutorial/_private/logic/LogicEntryHook.py`:

1. Add this import at the top of the file with the other imports:

```
from .LogicToAgentRpcCallExample import LogicToAgentRpcCallExample
```

2. Add this line just before the `logger.debug("Started")` line at the end of the `start()` method:

```
# Initialise and start the RPC for Logic Service
self._loadedObjects.append(LogicToAgentRpcCallExample().start())
```

The logic service will now call the RPC method on the agent when it starts.

Testing

1. Open a command window and run: `run_peek_logic_service`
2. Open a command window and run: `run_peek_agent_service`
3. Examine the logs of both command windows

`run_peek_logic_service` log example:

```
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.updateStatus
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.logic.controller.
↳MainController:Agent said : Agent RPC Example Started
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.addStringInt
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.updateStatus
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.logic.controller.
↳MainController:Agent said : Agent RPC Example Completed
```

`run_peek_agent_service` log example:

```
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.updateStatus
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↳tutorial._private.logic.agent_handlers.RpcForAgent.RpcForAgent.updateStatus
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.agent.
↳AgentToLogicRpcCallExample:seedInt = 5
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↳tutorial._private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.agent.
↳AgentToLogicRpcCallExample:seedInt = 12
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↳tutorial._private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.agent.
↳AgentToLogicRpcCallExample:seedInt = 19
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↳tutorial._private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.agent.
↳AgentToLogicRpcCallExample:seedInt = 26
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
```

(continues on next page)

(continued from previous page)

```

19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↳tutorial._private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.agent.
↳AgentToLogicRpcCallExample:seedInt = 33
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↳tutorial._private.logic.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.agent.
↳AgentToLogicRpcCallExample:seedInt = 40
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.addStringInt
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.agent.
↳AgentToLogicRpcCallExample:runWithInlineCallback finished
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↳tutorial._private.logic.agent_handlers.RpcForAgent.RpcForAgent.addStringInt
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↳private.logic.agent_handlers.RpcForAgent.RpcForAgent.updateStatus
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↳tutorial._private.logic.agent_handlers.RpcForAgent.RpcForAgent.updateStatus

```

3.1.18 Add Plugin Python API

Overview

Plugin APIs play a big part in the design and philosophy behind Peek.

Peeks philosophy is to create many small plugins that each do one job and do it well.

The idea being, once the plugin is written, you can leverage the functionality of that plugin with out worrying about the internal workings of it.

Another plugin benefit is to have many smaller code bases. It's easier for a rouge edit to be introduced into existing code with out strict review procedures. Separate code bases makes this impossible.

Same Service APIs Only

Plugins can only use the APIs of other plugins on the same service.

For example, the code from `peek_plugin_one` that runs on the Logic Service can only use the API published by the code in `peek_plugin_two` that runs on the Logic service.

What are APIs

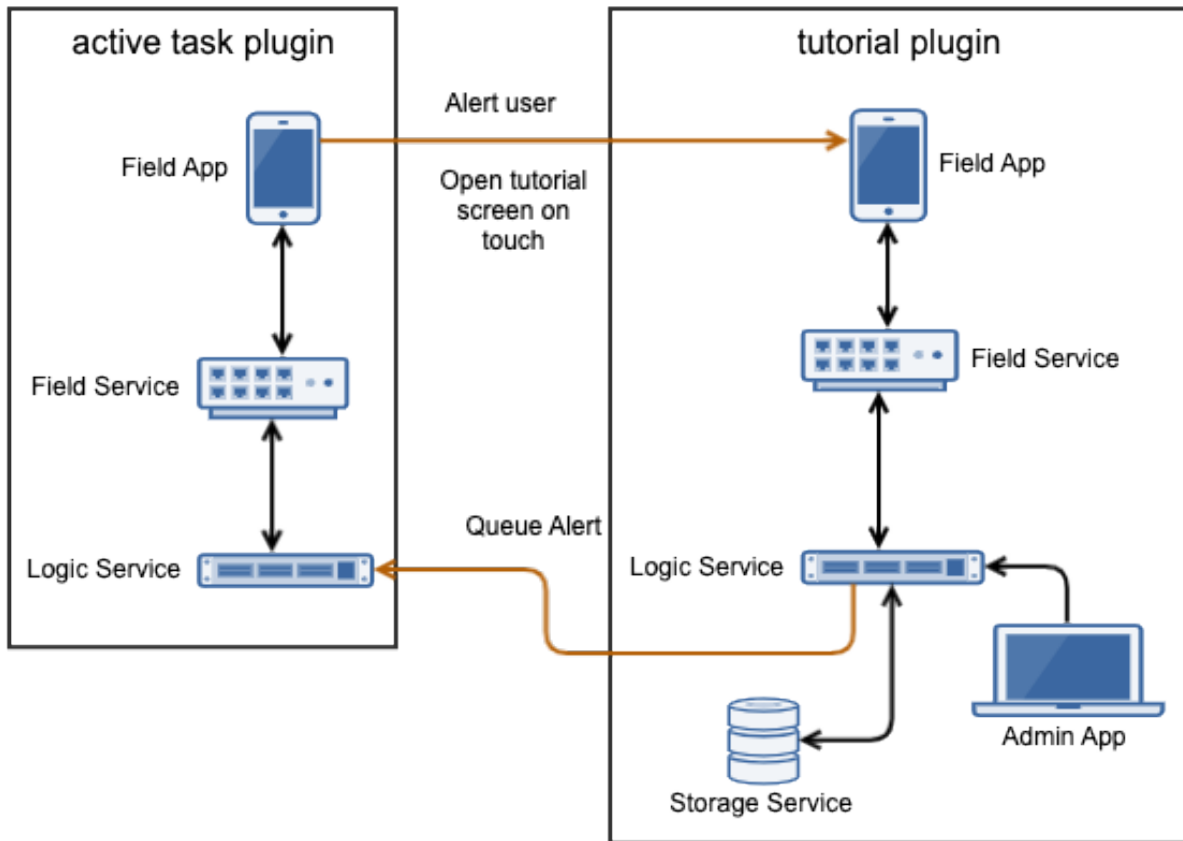
An API is an application programming interface, in python side Peek terms, it's an exposed abstract class or classes that other plugins can import and use. By "exposed" we mean, anything not under the `"_private"` package.

The Peek platform provides a method to grab a reference to another plugins exposed API object. The plugin grabbing another plugins API object reference can then call methods directly on it.

The ABC (Abstract Base Class) is purely for documentation purposes, and allows the real implementation to be hidden in the `_private` package.

In this example, we're going to expose an API for the Logic service in the `peek_plugin_tutorial` plugin.

We'll then get the API for the `peek_plugin_inbox` plugin and create a task.



Setup Logic API

In this section, we define an API on the Logic Service for the `peek_plugin_tutorial` plugin.

Add File `DoSomethingTuple.py`

File `DoSomethingTuple.py` defines a public tuple that will be returned from the API.

Create the file `peek_plugin_tutorial/tuples/DoSomethingTuple.py` and populate it with the following contents.

```
from peek_plugin_tutorial._private.PluginNames import tutorialTuplePrefix
from vortex.Tuple import Tuple, addTupleType, TupleField

@addTupleType
class DoSomethingTuple(Tuple):
    """ Do Something Tuple

    This tuple is publicly exposed and will be the result of the doSomething api call.
```

(continues on next page)

(continued from previous page)

```

"""
__tupleType__ = tutorialTuplePrefix + 'DoSomethingTuple'

#: The result of the doSomething
result = TupleField(defaultValue=dict)

```

Add Package logic

Have you ever wondered why everything so far has been under the `_private` package? It's about to make more sense.

The `peek_plugin_tutorial.logic` python package will contain the exposed API abstract classes.

Create the `peek_plugin_tutorial/logic` package, with the commands

```

mkdir peek_plugin_tutorial/logic
touch peek_plugin_tutorial/logic/__init__.py

```

Add File TutorialApiABC.py

File `TutorialApiABC.py` defines the interface of the API, including what should be detailed docstrings. It doesn't contain any implementation.

Create the file `peek_plugin_tutorial/logic/TutorialApiABC.py` and populate it with the following contents.

```

from abc import ABCMeta, abstractmethod

from peek_plugin_tutorial.tuples.DoSomethingTuple import DoSomethingTuple

class TutorialApiABC(metaclass=ABCMeta):

    @abstractmethod
    def doSomethingGood(self, somethingsDescription:str) -> DoSomethingTuple:
        """ Add a New Task

        Add a new task to the users device.

        :param somethingsDescription: An arbitrary string
        :return: The computed result contained in a DoSomethingTuple tuple

        """

```

Add File TutorialApi.py

File `TutorialApi.py` is the implementation of the API. An instance of this class will be passed to other APIs when they ask for it.

Create the file `peek_plugin_tutorial/_private/logic/TutorialApi.py` and populate it with the following contents.

```
from peek_plugin_tutorial._private.logic.controller.MainController import MainController
↳MainController
from peek_plugin_tutorial.logic.TutorialApiABC import TutorialApiABC
from peek_plugin_tutorial.tuples.DoSomethingTuple import DoSomethingTuple

class TutorialApi(TutorialApiABC):
    def __init__(self, mainController: MainController):
        self._mainController = mainController

    def doSomethingGood(self, somethingsDescription: str) -> DoSomethingTuple:
        """ Do Something Good

        Add a new task to the users device.

        :param somethingsDescription: An arbitrary string

        """

        # Here we could pass on the request to the self._mainController if we wanted.
        # EG self._mainController.somethingCalled(somethingsDescription)

        return DoSomethingTuple(result="SUCCESS : " + somethingsDescription)

    def shutdown(self):
        pass
```

Edit File `LogicEntryHook.py`

We need to update `LogicEntryHook.py`, to initialise the API object.

Edit the file `peek_plugin_tutorial/_private/logic/LogicEntryHook.py`:

1. Add this import at the top of the file with the other imports:

```
from .TutorialApi import TutorialApi
```

2. Add this line at the end of the `__init__(...):` method:

```
self._api = None
```

3. Add this line just before the `logger.debug("Started")` line at the end of the `start()` method:

```
# Initialise the API object that will be shared with other plugins
self._api = TutorialApi(mainController)
self._loadedObjects.append(self._api)
```

4. Add this line just before the `logger.debug("Stopped")` line at the end of the `stop()` method:

```
self._api = None
```

5. Add this method to end of the `LogicEntryHook` class:

```
@property
def publishedLogicApi(self) -> object:
    """ Published Logic Service API

    :return class that implements the API that can be used by other Plugins on_
    ↪this
    platform service.
    """
    return self._api
```

The API is now accessible from other plugins.

Use Logic Service API

In this section we'll get a reference to the Peek Plugin Inbox API and then create a task on the Field or Office UI.

Note: In order to use this example, you will need to have the `peek_core_user` plugin installed and enabled in the Field, Office, and Logic services, via their config.json files.

The user plugin is public, it can be installed with `pip install peek-core-user`.

Note: In order to use this example, you will need to have the `peek_plugin_inbox` plugin installed and enabled in the Field, Office, and Logic services, via their config.json files.

The peek inbox plugin is public, it can be installed with `pip install peek_plugin_inbox`.

Add File `ExampleUseTaskApi.py`

File `ExampleUseTaskApi.py` contains the code that uses the Peek Inbox Tasks API.

Create the file `peek_plugin_tutorial/_private/logic/ExampleUseTaskApi.py` and populate it with the following contents.

Replace the "userId" with your user id.

```
import logging
import pytz
from datetime import datetime

from twisted.internet import reactor
from twisted.internet.defer import inlineCallbacks

from peek_plugin_inbox.logic.InboxApiABC import InboxApiABC, NewTask
from peek_plugin_tutorial._private.logic.service.controller.MainController import_
↪MainController
from peek_plugin_tutorial._private.PluginNames import tutorialPluginName

logger = logging.getLogger(__name__)
```

(continues on next page)

(continued from previous page)

```
class ExampleUseTaskApi:
    def __init__(self, mainController: MainController, inboxApi: InboxApiABC):
        self._mainController = mainController
        self._inboxApi = inboxApi

    def start(self):
        reactor.callLater(1, self.sendTask)
        return self

    @inlineCallbacks
    def sendTask(self):
        # First, create the task
        newTask = NewTask(
            pluginName=tutorialPluginName,
            uniqueId=str(datetime.now(pytz.utc)),
            userId="userId", # <----- Set to your user id
            title="A task from tutorial plugin",
            description="Tutorials task description",
            routePath="/peek_plugin_tutorial",
            autoDelete=NewTask.AUTO_DELETE_ON_SELECT,
            overwriteExisting=True,
            notificationRequiredFlags=NewTask.NOTIFY_BY_DEVICE_SOUND
                                   | NewTask.NOTIFY_BY_EMAIL
        )

        # Now send the task via the inbox tasks API
        yield self._inboxApi.addTask(newTask)

        logger.debug("Task Sent")

    def shutdown(self):
        pass
```

Edit File LogicEntryHook.py

We need to update LogicEntryHook.py, to initialise the example code

Edit the file peek_plugin_tutorial/_private/logic/LogicEntryHook.py:

1. Add this import at the top of the file with the other imports:

```
from peek_plugin_inbox.logic.InboxApiABC import InboxApiABC
from .ExampleUseTaskApi import ExampleUseTaskApi
```

2. Add this line just before the logger.debug("Started") line at the end of the start() method:

```
# Get a reference for the Inbox Task
inboxApi = self.platform.getOtherPluginApi("peek_plugin_inbox")
assert isinstance(inboxApi, InboxApiABC), "Wrong inboxApi"
# Initialise the example code that will send the test task
self._loadedObjects.append(
    ExampleUseTaskApi(mainController, inboxApi).start()
)
```

Testing

1. Open Field web app
2. Tap Task icon located in the top right corner
3. You will see the task in the list

3.1.19 Add Plugin TypeScript APIs (TODO)

Overview

This document will describe how to use the APIs between plugins running on the:

- Field
- Office
- and Admin services

These services all run TypeScript + Angular, the integrations are provided by the standard Angular services mechanisms.

How To

For a plugin to publish an API, Create an Angular Service.

For a plugin to use another plugins API, Use that service in the constructor of your Angular service, component or module.

Warning: Be careful with singleton services, adding it to multiple provides will cause the service to be created again instead of looking for a provider in the parent.

That's basically how this will work. Examples to come at a later date.

3.1.20 Use Plugin APIs (TODO)

Overview

This doc describes how to retrieve and use APIs from other plugins.

3.1.21 Add Worker Service

Outline

In this document, we setup the Worker service and submit a job from the Logic service. The Logic service will send a random number to worker and, the worker inverts the number and sends it back.

Add Package `_private/worker`

Create directory `peek_plugin_tutorial/_private/worker`

Create an empty package file in the worker directory: `peek_plugin_tutorial/_private/worker/__init__.py`

Commands:

```
mkdir peek_plugin_tutorial/_private/worker
touch peek_plugin_tutorial/_private/worker/__init__.py
```

Add File `WorkerEntryHook.py`

Create the file `peek_plugin_tutorial/_private/worker/EntryHook.py` and populate it with the following contents.

```
import logging
from peek_plugin_base.worker.WorkerEntryHookABC import WorkerEntryHookABC
from peek_plugin_tutorial._private.worker.tasks import RandomNumber

logger = logging.getLogger(__name__)

class WorkerEntryHook(PluginWorkerEntryHookABC):
    def __init__(self, *args, **kwargs):
        """ Constructor """
        # Call the base classes constructor
        WorkerEntryHookABC.__init__(self, *args, **kwargs)

        #: Loaded Objects, This is a list of all objects created when we start
        self._loadedObjects = []

    def load(self) -> None:
        """ Load

        This will be called when the plugin is loaded, just after the db is migrated.
        Place any custom initialiaction steps here.

        """
        logger.debug("Loaded")

    def start(self):
        """ Load

        This will be called when the plugin is loaded, just after the db is migrated.
        Place any custom initialiaction steps here.

        """
        logger.debug("Started")

    def stop(self):
        """ Stop

        This method is called by the platform to tell the peek app to shutdown and
↪ stop
        everything it's doing
```

(continues on next page)

(continued from previous page)

```

    """
    # Shutdown and dereference all objects we constructed when we started
    while self._loadedObjects:
        self._loadedObjects.pop().shutdown()

    logger.debug("Stopped")

    def unload(self):
        """Unload

        This method is called after stop is called, to unload any last resources
        before the PLUGIN is unlinked from the platform

        """
        logger.debug("Unloaded")

    @property
    def celeryAppIncludes(self):
        return [RandomNumber.__name__]

```

Add Package `_private/worker/tasks`

Create directory `_private/worker/tasks`

Create an empty package file in the tasks directory, `peek_plugin_tutorial/_private/worker/tasks/__init__.py`

Commands:

```

mkdir -p peek_plugin_tutorial/_private/worker/tasks
touch peek_plugin_tutorial/_private/worker/tasks/__init__.py

```

Add File `RandomNumber.py`

Create the file `peek_plugin_tutorial/_private/worker/tasks/RandomNumber.py` and populate it with the following contents. This worker returns the negative number for the given positive number

```

import logging
from random import randint
from txcelery.defer import DeferrableTask
from peek_plugin_base.worker.CeleryApp import celeryApp

logger = logging.getLogger(__name__)

@DeferrableTask
@celeryApp.task(bind=True)
def pickRandomNumber(self, item: int) -> int:
    """
    Returns random integer between 1 to 1000
    """
    return int(item) * -1

```

Edit peek_plugin_tutorial/__init__.py

Edit the file peek_plugin_tutorial/__init__.py, and add the following:

```
from peek_plugin_base.worker.PluginWorkerEntryHookABC import PluginWorkerEntryHookABC
from typing import Type

def peekWorkerEntryHook() -> Type[PluginWorkerEntryHookABC]:
    from ._private.worker.WorkerEntryHook import WorkerEntryHook
    return WorkerEntryHook
```

Edit plugin_package.json

Edit the file peek_plugin_tutorial/plugin_package.json:

1. Add “**worker**” to the requiresServices section so it looks like

```
"requiresServices": [
    "worker"
]
```

2. Add the **worker** section after **requiresServices** section:

```
"worker": {
}
```

3. Ensure your JSON is still valid (Your IDE may help here)

Here is an example

```
{
  "plugin": {
    ...
  },
  "requiresServices": [
    "worker"
  ],
  "worker": {
  }
}
```

The plugin should now be ready for the worker to load.

Running on the Worker Service

Edit ~/peek-worker-service.home/config.json:

1. Ensure **logging.level** is set to “**DEBUG**”
2. Add “**peek_plugin_tutorial**” to the **plugin.enabled** array

Note: It would be helpful if this is the only plugin enabled at this point.

It should somthing like this:


```
{
    ...
    "logging": {
        "level": "DEBUG"
    },
    ...
    "plugin": {
        "enabled": [
            "peek_plugin_tutorial"
        ],
        ...
    },
    ...
}
```

Note: This file is created in *Administration*

You can now run the peek worker, you should see your plugin load.

```
peek@peek:~$ run_peek_worker_service
...
DEBUG peek_plugin_tutorial._private.worker.WorkerEntryHook:Loaded
DEBUG peek_plugin_tutorial._private.worker.WorkerEntryHook:Started
...
```

Push work from logic to worker service

Note: Ensure rabbitmq and redis services are running

Create `peek_plugin_tutorial/_private/logic/controller/RandomNumberWorkerController.py` with below content:

```
import logging
from twisted.internet import task, reactor, defer
from twisted.internet.defer import inlineCallbacks
from vortex.DeferUtil import deferToThreadWrapWithLogger, vortexLogFailure
from datetime import datetime
from random import randint
import pytz

logger = logging.getLogger(__name__)

class RandomNumberWorkerController:
    """
        Random Number Generator
        Generates random number on worker periodically
    """

    PERIOD = 5
```

(continues on next page)

(continued from previous page)

```
TASK_TIMEOUT = 60.0

def __init__(self):
    self._pollLoopingCall = task.LoopingCall(self._poll)

def start(self):
    d = self._pollLoopingCall.start(self.PERIOD, now=False)
    d.addCallbacks(self._timerCallback, self._timerErrback)

def _timerErrback(self, failure):
    vortexLogFailure(failure, logger)

def _timerCallback(self, _):
    logger.info("Time executed successfully")

def stop(self):
    if self._pollLoopingCall.running:
        self._pollLoopingCall.stop()

def shutdown(self):
    self.stop()

@inlineCallbacks
def _poll(self):
    # Send the tasks to the peek worker
    start = randint(1, 1000)
    try:
        result = yield self._sendToWorker(start)
    catch Exception as e:
        logger.exception(e)

@inlineCallbacks
def _sendToWorker(self, item):
    from peek_plugin_tutorial._private.worker.tasks.RandomNumber import _
    ↪pickRandomNumber
    startTime = datetime.now(pytz.utc)

    try:
        d = pickRandomNumber.delay(item)
        d.addTimeout(self.TASK_TIMEOUT, reactor)
        randomNumber = yield d
        logger.debug("Time Taken = %s, Random Number: %s" % (datetime.now(pytz.
    ↪utc) - startTime, randomNumber))
    except Exception as e:
        logger.debug(" RandomNumber task failed : %s", str(e))
```

Edit peek_plugin_tutorial/_private/logic/LogicEntryHook.py:

1. Add the following imports at the top of the file with the other imports:

```
from peek_plugin_base.logic.PluginLogicWorkerEntryHookABC import _
    ↪PluginLogicWorkerEntryHookABC
from peek_plugin_tutorial._private.logic.controller.RandomNumberWorkerController _
    ↪import RandomNumberWorkerController
```

2. Add PluginLogicWorkerEntryHookABC to list of inherited class:

```
class LogicWorkerEntryHook(PluginLogicWorkerEntryHookABC, ...):
```

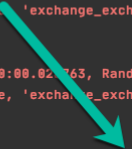
3. Add this line just before the `logger.debug("Started")` line at the end of the `start()` method:

```
randomNumberController = RandomNumberWorkerController()
self._loadedObjects.append(randomNumberController)
randomNumberController.start()
```

Run `run_peek_logic_service`

You can now run the peek logic service, you should see output like below, showing the :

```
29-Apr-2020 15:47:30 DEBUG txcelery.defer:Closing connections for thread 123145659813888
29-Apr-2020 15:47:30 DEBUG txcelery.defer:Closing connections for thread 123145481138176
29-Apr-2020 15:47:30 DEBUG amqp:Closed channel #1
29-Apr-2020 15:47:30 DEBUG amqp:Closed channel #1
29-Apr-2020 15:47:30 DEBUG amqp:Start from server, version: 0.9, properties: {'capabilities': {'publisher_confirms': True, 'exchange_exchange_bindings': Tr
29-Apr-2020 15:47:30 DEBUG amqp:using channel_id: 1
29-Apr-2020 15:47:30 DEBUG amqp:Channel open
29-Apr-2020 15:47:30 DEBUG peek_plugin_tutorial._private.server.controller.RandomNumberWorkerController:Time Taken = 0:00:00.02763, Random Number: -1
29-Apr-2020 15:47:30 DEBUG amqp:Start from server, version: 0.9, properties: {'capabilities': {'publisher_confirms': True, 'exchange_exchange_bindings': Tr
29-Apr-2020 15:47:35 DEBUG amqp:using channel_id: 1
29-Apr-2020 15:47:35 DEBUG amqp:Channel open
29-Apr-2020 15:47:35 DEBUG peek_plugin_tutorial._private.server.controller.RandomNumberWorkerController:Time Taken = 0:00:00.029364, Random Number: -516
29-Apr-2020 15:47:40 DEBUG amqp:Start from server, version: 0.9, properties: {'capabilities': {'publisher_confirms': True, 'exchange_exchange_bindings': Tr
29-Apr-2020 15:47:40 DEBUG amqp:using channel_id: 1
29-Apr-2020 15:47:40 DEBUG amqp:Channel open
29-Apr-2020 15:47:40 DEBUG peek_plugin_tutorial._private.server.controller.RandomNumberWorkerController:Time Taken = 0:00:00.030222, Random Number: -964
```



3.1.22 Challenges

This document is designed to test your knowledge about the Peek platform.

Challenge #1: Lifecycle Methods

This challenge will test your knowledge of lifecycle methods within Peek services.

Tasks:

- Log the current time when the peek logic service is in its *start* lifecycle.

The following module will be required:

```
import datetime
```

The output should resemble this:

```
10-Aug-2020 10:08:44 DEBUG peek_plugin_tutorial._private.logic.LogicEntryHook:2020-08-
↪10 10:08:44.223694
10-Aug-2020 10:08:44 DEBUG peek_plugin_tutorial._private.logic.LogicEntryHook:Started
```

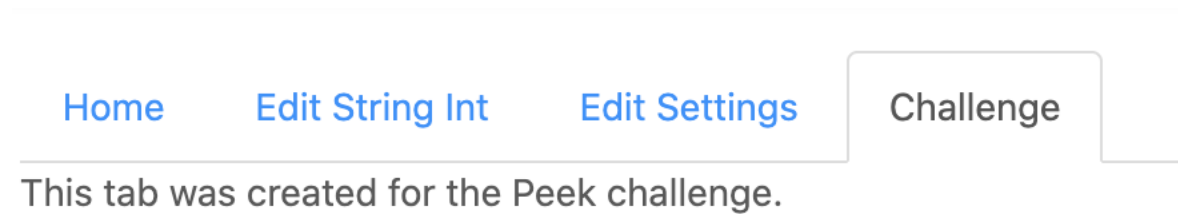
Challenge #2: Admin Plugin Tab

This challenge will test your knowledge of the web interfaces within Peek services.

Tasks:

- Develop a new tab in the “Tutorial Plugins” page within the admin Peek site.

The output should resemble this:



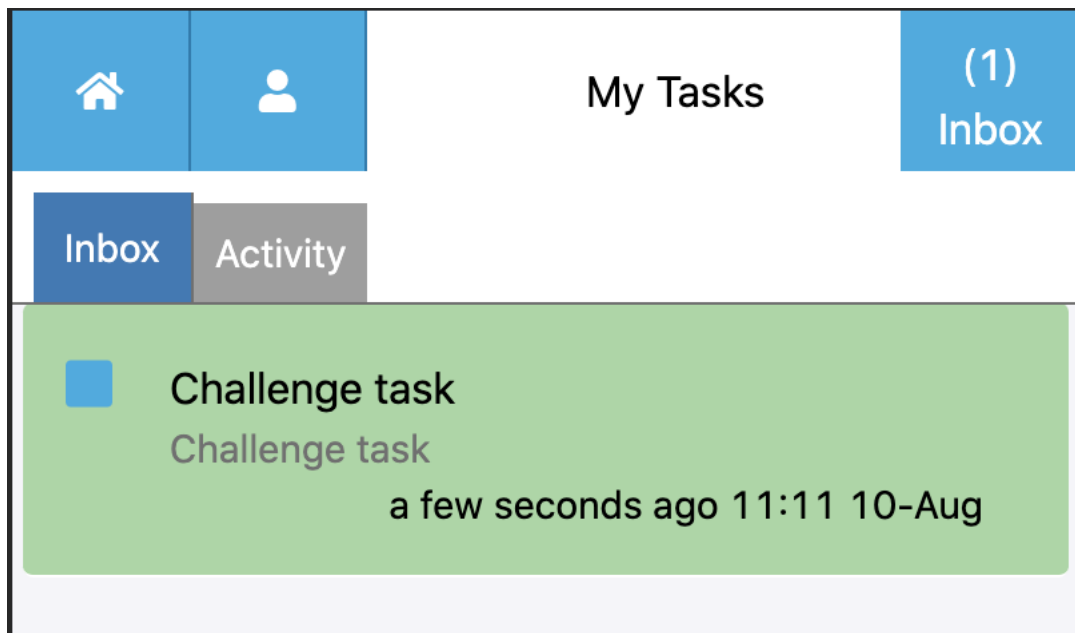
Challenge #3: Field/Office Tasks

This challenge will test your understanding of the various components located within the admin Peek site, and how they interact with other Peek services.

Tasks:

- Send a task to the field / office service from the admin Peek site.

The output should resemble this:



Challenge #4: VortexJs Tuple Actions

This challenge will test your understanding of how tuple actions work, and how they are used by the various Peek services.

Tasks:

- Create an action that doubles the current int value displayed in the StringIntComponent.
- Initiate the action via a button click on the field app.
- Display the outcome on the field app.

The output should resemble this:

[Back to Main](#)

String	Int	
my string	36	Toggle Caps Increment Int Decrement Int Double Int

Challenge #5: Worker Service

This challenge will test your understanding of how the worker service and logic service communicate with one another.

Tasks:

- Create a new task named “SpecificNumber” for the worker service to complete.
- Make the task log the number “24” and the time taken to complete the task every 5 seconds.

The output should resemble this:

```
10-Aug-2020 14:10:31 DEBUG celery.worker.request:Task accepted:peek_plugin_tutorial._
↳private.worker.tasks.SpecificNumber.pickSpecificNumber[fc6ee8bf-1e11-4481-b84b-
↳66cf7e4f197a] pid:82053
10-Aug-2020 14:10:31 INFO celery.app.trace:Task peek_plugin_tutorial._private.worker.
↳tasks.SpecificNumber.pickSpecificNumber[fc6ee8bf-1e11-4481-b84b-66cf7e4f197a]_
↳succeeded in 0.0007026020030025393s: 24
```

3.2 Twisted Python Tutorial

3.2.1 Understanding Twisted

Introduction to Twisted

This tutorial is here to help new users to become familiar with using the twisted library to write asynchronous programs in Python.

Opening Assumptions

This tutorial assumes that you:

1. Have set up a Python development environment.
2. Have at least a basic level of familiarity with the Python programming language and Object Oriented programming.

What is Twisted?

Twisted is an **asynchronous** networking framework for Python, as opposed to synchronous.

Synchronous computing is when each event happens after the preceding event finishes. Thing 1 always follows Thing 2. But what happens if Thing 1 takes 10 seconds to complete and 7 seconds of that is spent waiting for a response from something else, as is often the case in networking? Thing 2 has to wait. In some situations this is acceptable, but in others we need for Thing 2 to happen as soon as possible.

This is where **asynchronous** frameworks like Twisted become useful. With Twisted's ability to **defer** tasks until they are ready, it is entirely possible for a chain of events to look like this:

- Thing 1 fires
- Thing 2 fires
- Thing 2 returns
- Thing 1 returns

As you can imagine, this has the potential to save a lot of time, especially in situations where users absolutely cannot be waiting longer than necessary for information.

So, how do we accomplish this with Twisted? Strap in and find out.

Installing Twisted

In order to use Twisted, you will need to install it. We will be using pip to do so. Start by opening a terminal in your project directory on Mac or Linux and entering:

```
pip install twisted
```

This will download and install twisted. When this completes, you will be able to import twisted inside of your project. Now we can begin to use Twisted to write programs.

Unit Tests

For the sake of brevity, many of the examples provided in this tutorial will take place inside of unit-tests, which will automatically take care of instantiating, starting, and stopping Twisted's reactor. Please note this when implementing these examples inside of your own programs.

Our First Twisted Programs

In this document, we will introduce the concept of a Twisted reactor, implement a simple program using a reactor, and then implement a simple webserver using a reactor and some of the basic networking classes available in the Twisted module.

Twisted Reactor Overview

The Twisted `reactor` is an implementation of the [Reactor Design Pattern](#). It manages an 'event loop', organizing which processes are currently able to run, and which are being **Deferred** to ensure that while the `reactor` is waiting on one thing, other things are still being completed.

We will explore the ways that the reactor can be used by:

1. Setting up and then explain a simple Twisted program, to illustrate this functionality.
2. Creating a simple webserver which makes use of a Twisted reactor below.

A simple Twisted program

The absolute simplest Twisted program you could possibly make, is as follows:

```
from twisted.internet import reactor
reactor.run()
```

This starts a Twisted reactor. In simple terms, the reactor is a loop that will wait for work to become assigned to it, and then do it.

If at any point during the work, it finds that it is waiting on something that is **Deferred**, it will move on to the next thing in its queue. When the **Deferred** object finally resolves into something, it will be notified and *react* to the change.

This way, it never leaves any work waiting while something out of its control is being completed.

Of course, right now, our reactor has not been given any work at all, so it will simply run silently until the process is halted. If you have not already done so, close it down now.

Now that that you have implemented a simple twisted reactor program, we can move on to creating something a little more complex.

A simple Twisted server

Now, we are going to use Twisted to build a simple web server.

Copy and paste the following code into a new project:

```
from twisted.internet import reactor
from twisted.internet.protocol import Protocol
from twisted.internet.protocol import ServerFactory
from twisted.internet.endpoints import TCP4ServerEndpoint

class ServerProtocol(Protocol):

    def connectionMade(self):
        print("New Connection")
        self.transport.write(b"Hello Twisted")
        self.transportloseConnection()

class TutorialServerFactory(ServerFactory):

    def buildProtocol(self, addr):
        print(addr)
        return ServerProtocol()

if __name__ == '__main__':
    endpoint = TCP4ServerEndpoint(reactor, 2000)
    endpoint.listen(TutorialServerFactory())
    reactor.run()
```

Run the above code block. While it is running, open your web browser and navigate to:

<http://localhost:2000/>

You should see a message that says “Hello Twisted”, indicating that your Twisted web server is up and running.

But how did we get here?

Let’s break it down step by step.

```
from twisted.internet import reactor
from twisted.internet.protocol import Protocol
from twisted.internet.protocol import ServerFactory
from twisted.internet.endpoints import TCP4ServerEndpoint
```

These lines all ensure that the right libraries are imported from Twisted to be used in our code. First comes the reactor to manage our simple event loop, then protocols and endpoints for communicating through a network. It’s OK if you don’t understand these yet, as you use the various Twisted libraries more, you will become familiar with them.

```
class TutorialServerFactory(ServerFactory):
    def buildProtocol(self, addr):
        print(addr)
        return ServerProtocol()
```

The `TutorialServerFactory` class inherits from `Protocol.ServerFactory`.

When a `ServerFactory` (or any subclass of `Protocol`) is initialized, it calls `buildProtocol`, which usually creates an instance of a `Protocol` object for us.

By defining `buildProtocol` locally, We have overridden it to have it to print out the address that it is being connected to from and return an instance of our own `ServerProtocol` class.

```
class ServerProtocol(Protocol):
    def connectionMade(self):
        print("New Connection")
        self.transport.write(b"Hello Twisted")
        self.transportloseConnection()
```

The `ServerProtocol` class inherits from `protocol.Protocol`. The `Protocol` class provides Twisted’s networking protocols and has several inbuilt functions that fire at predefined times. The one we take advantage of in our example is `connectionMade()` which runs every time someone connects (or reconnects) to the server.

There are other, similar functions within `Protocol`, like `connectionLost()`, but we will not be using them for now.

As you can see, every time a connection is made, our `Server` class will fire `connectionMade()` which will print “New Connection” to the console, and then call `self.transport.write()`, which sends the message you see in your browser window, before finally dropping the connection with `self.transportloseConnection()`.

This part is only ever run if the script is run directly, which in our case it is.

```
if __name__ == '__main__':
    endpoint = TCP4ServerEndpoint(reactor, 2000)
    endpoint.listen(TutorialServerFactory())
    reactor.run()
```


This code block:

1. Creates a `TCP4ServerEndpoint` which uses Twisted's `reactor` to keep track of its event loop, and listens on port 2000.
2. Defines the protocol family that it will use to listen to the TCP endpoint (our `:code:TutorialServerFactory`).
3. Starts the `reactor`, which waits until our endpoints schedule something for it to do, and then it does them.

Although it implements the Twisted reactor, right now our server is only behaving synchronously. This is because we have not attempted to actually do anything asynchronously yet.

Next, we will introduce the concept of a **Deferred** and implement some **asynchronous** functions.

The Twisted Reactor Explained

Twisted's reactor is the object that controls the main event loop. It allows data to be processed **asynchronously** and it loosely follows the titular [Reactor Design Pattern](#).

What are the benefits

The Twisted reactor has two main benefits.

1. Asynchronous computing.
2. It implements a [select loop](#) and not a [busy loop](#). This means that while idle, the reactor will not consume CPU by constantly checking to see if there is something to do.

FAQ

How do I instantiate the Twisted reactor?

You **don't** ever need to instantiate it, instead just import it. The twisted reactor is a [Singleton](#). This means there is only ever one instance. If an instance of it does not already exist, then it is created at the time that it is imported.

How do I start it?

You can start the Twisted reactor by calling:

```
reactor.run()
```

Before this point it will still exist, but will not act.

Similarly, you can stop the reactor by calling:

```
reactor.stop()
```

Which will shut the reactor down. Once it is shut down, it will no longer be processing functions, so restarting it programmatically from inside will be impossible.

Is there more than one implementation of the Twisted reactor?

Yes, there are multiple implementations of the Twisted reactor that you can import depending on what you require it to do.

You can find the various implementations in `twisted.internet`. The filenames follows the format `*reactor`.

By default, importing reactor will import an **EPollReactor** for Linux-based operating systems, and either **PollReactor** or **SelectReactor** for other operating systems.

3.2.2 Understanding Deferreds

Introduction to Deferred

In this section we will examine a brief overview of what Twisted's Deferred object is and what it does, before implementing it in a way that shows how it can be used to gain a performance advantage over synchronous code.

What is a Deferred?

A Deferred is an object that represents a promise that at some point it will resolve into another kind of object. Deferred's allow the reactor to avoid spending time waiting for something outside of its control to complete.

A Synchronous Problem

Examine the following **synchronous** code block:

```
import time

def printMe(value):
    print(value)

def waitFive(name) -> str:
    value = name + " took 5 seconds"
    time.sleep(5)
    return value

def waitTen(name) -> str:
    value = name + " took 10 seconds"
    time.sleep(10)
    return value

print("Starting: ")

# Define two variables that will take a while to resolve
slow = waitTen('slow')
fast = waitFive('fast')

printMe(slow)
printMe(fast)
```

This represents a standard, synchronous situation. “slow” is assigned the value returned by waitTen() and then “fast” is assigned the value returned by waitFive(). Things always happen after the previous thing finishes and in total, the program takes 15 seconds to complete. The vast majority of that time is spent waiting.

A Deferred Solution

Now, open your IDE and insert the following **asynchronous** code block:

```
from twisted.internet import reactor, defer

def printMe(value):
    print(value)
    return True

def waitFive(name) -> defer.Deferred:
    value = defer.Deferred()
    reactor.callLater(5, value.callback, name + " took 5 seconds")
    return value

def waitTen(name) -> defer.Deferred:
    value = defer.Deferred()
    reactor.callLater(10, value.callback, name + " took 10 seconds")
    return value

print("Starting: ")

# Define two variables that will take a while to resolve
slow = waitTen("'slow'")
fast = waitFive("'fast'")

# Show that they are being Deferred
print("slow is: ", slow)
print("fast is: ", fast)

# Give them callbacks
slow.addCallback(printMe)
fast.addCallback(printMe)

reactor.run()
```

Run it and after a few seconds you should see the following result:

```
Starting:
slow is:  <Deferred at 0x7f57b1833340>
fast is:  <Deferred at 0x7f57b16eee50>
'fast' took 5 seconds
'slow' took 10 seconds
```

As you can see, both variables were assigned Deferred objects immediately after their creation the variable “fast” was printed before “slow” this time, and as a result, the values are returned in 10 seconds. We saved 5 seconds in execution time! But how did we get here?

The first thing you may have noticed is that this time in addition to the reactor, we also imported `defer` from `twisted.internet`.

```
from twisted.internet import reactor, defer
```

This allows us to use Twisted's Deferred objects.

Take a look at one of the wait functions.

```
def waitTen() -> defer.Deferred:
    value = defer.Deferred()
    reactor.callLater(10, value.callback, "I took 10 seconds")
    return value
```

Instead of a string like in our synchronous example, `value` is assigned a `defer.Deferred()` object. This allows us to return a placeholder object immediately, so that we can keep processing while the actual value is determined and passed back to it. Then, to simulate a long wait, the reactor is asked via `callLater` to come back to our Deferred in 10 seconds, assigning it the value "I took 10 seconds" and starting its **callback chain**. As soon as this request is made, `value` is returned and assigned to "slow".

```
slow = waitTen()
fast = waitFive()

...
```

```
slow.addCallback(printMe)
fast.addCallback(printMe)
```

Because "slow" is immediately assigned a value, we are able to move forward and assign "fast", to the output of `waitFive()`, which returns a Deferred in exactly the same way, only after 5 seconds instead of 10.

5 and 10 seconds later respectively, the reactor calls the Deferred's `callback(any)` function, passing each of them a string and triggering their waiting callbacks, which in this case are both "printMe()". Because "fast" finished first, it was printed first, and "slow", although it was started first, was printed after.

Deferred Inline Callbacks

As we learned in our first Twisted asynchronous tutorial, Deferreds are happy to step aside while they are waiting for something and allow other things to execute out of order. This is not always desired.

Twisted's `@inlineCallback` wrapper allows the user to pause the execution of anything which results in a deferred and wait for the deferred to resolve using the `yield` keyword.

When to use it

When the *final* resolved value of something is required in order to safely continue. [These are used frequently in Peek.](#)

How to use it

Below, we create a function that uses `@inlineCallbacks` and a function that does not:

```

from twisted.internet.defer import inlineCallbacks
from twisted.internet.task import deferLater
from twisted.internet import reactor

from twisted.trial import unittest

class InlineExampleTest(unittest.TestCase):

    @inlineCallbacks
    def testWithInlineCallback(self):

        d = yield deferLater(reactor, 2, lambda: True)
        print(d)
        print("x")
        return d

    def testWithoutInlineCallback(self):
        d = deferLater(reactor, 2, lambda: True)
        d.addCallback(print)
        print(d)
        print("x")
        return d

```

When `testWithInlineCallback` is run, it should print `True`, followed by `x`, indicating that execution was put on hold while `d` resolved, resulting in an output like this:

```

True
x

Ran 1 test in 2.007s

OK

```

On the other hand, `testWithoutInlineCallback` should immediately print a Deferred object, then `x`, and then finally `True`, indicating that progress was able to continue before it resolved, with an output similar to this:

```

<Deferred at 0x10f300c88>
x
True

Ran 1 test in 2.007s

OK

```

Deferred Callback Propagation

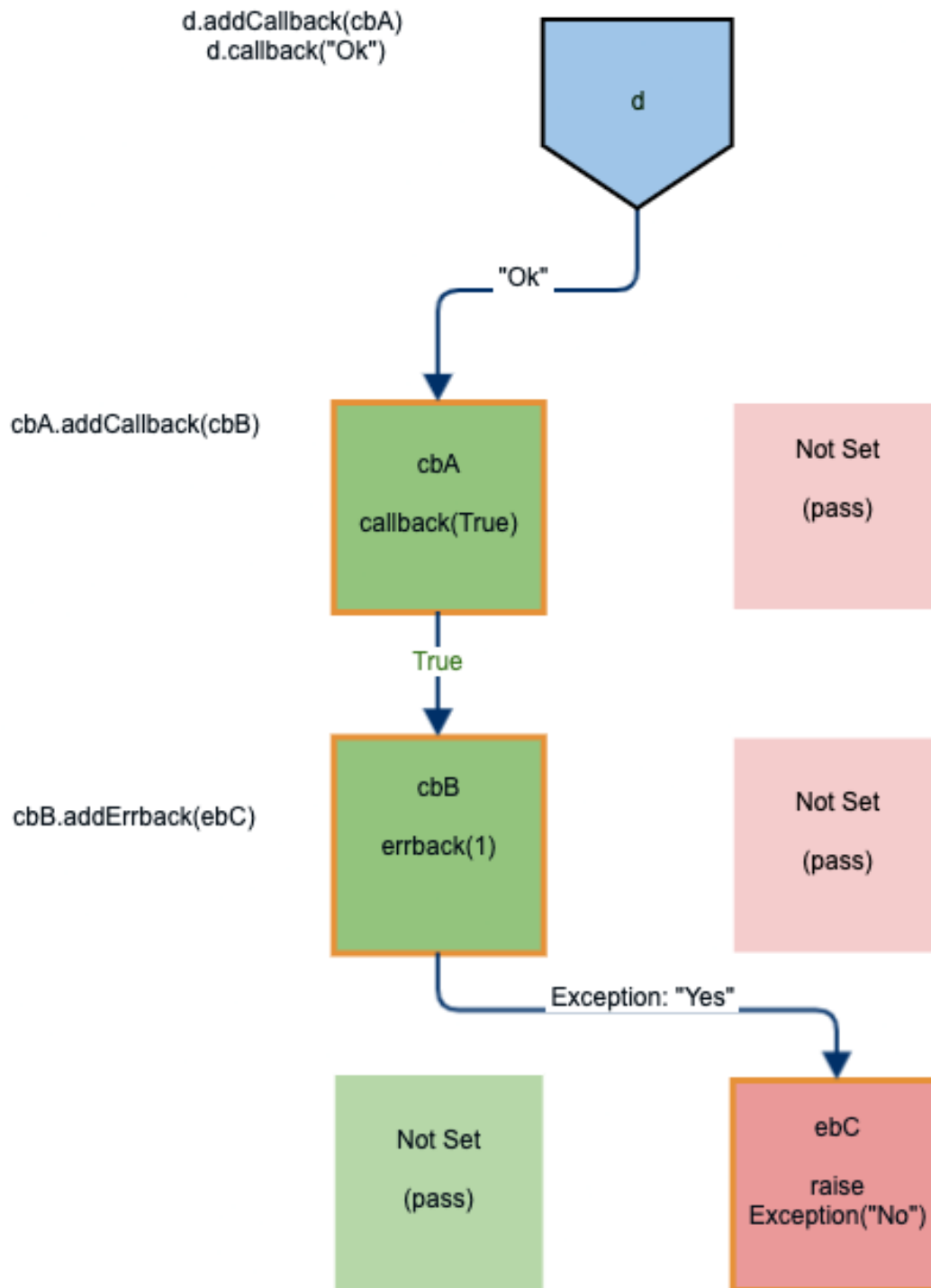
Before a Deferred has resolved, it can have a number of callbacks and an errbacks added to it, that will tell it what to do in the case of a failure.

In many cases, these callbacks or errbacks can also return a Deferred, which may in turn return yet more Deferreds, each with their own callbacks and errbacks.

Callbacks may even be called by errbacks and vice versa, all the way along the chain.

At the end of this tangled (dare I say, ‘twisted’) process, the final returned value of the final callback or errback in the chain is the value that the initial Deferred will be passed.

While the callback chain sounds like a difficult process to grasp, it is usually very straightforward. For a more visual explanation, consider the following diagram:



If a chain of callbacks goes `d -> cbA -> ebB`, the final value of A will be the value returned by `ebB`.

In code, the above diagram would look similar to this:

```
def testCallbackPropagationExample(self):
    def cbA(data):
        d = defer.Deferred()
        d.addCallback(cbB)
        d.callback(True)
        return d

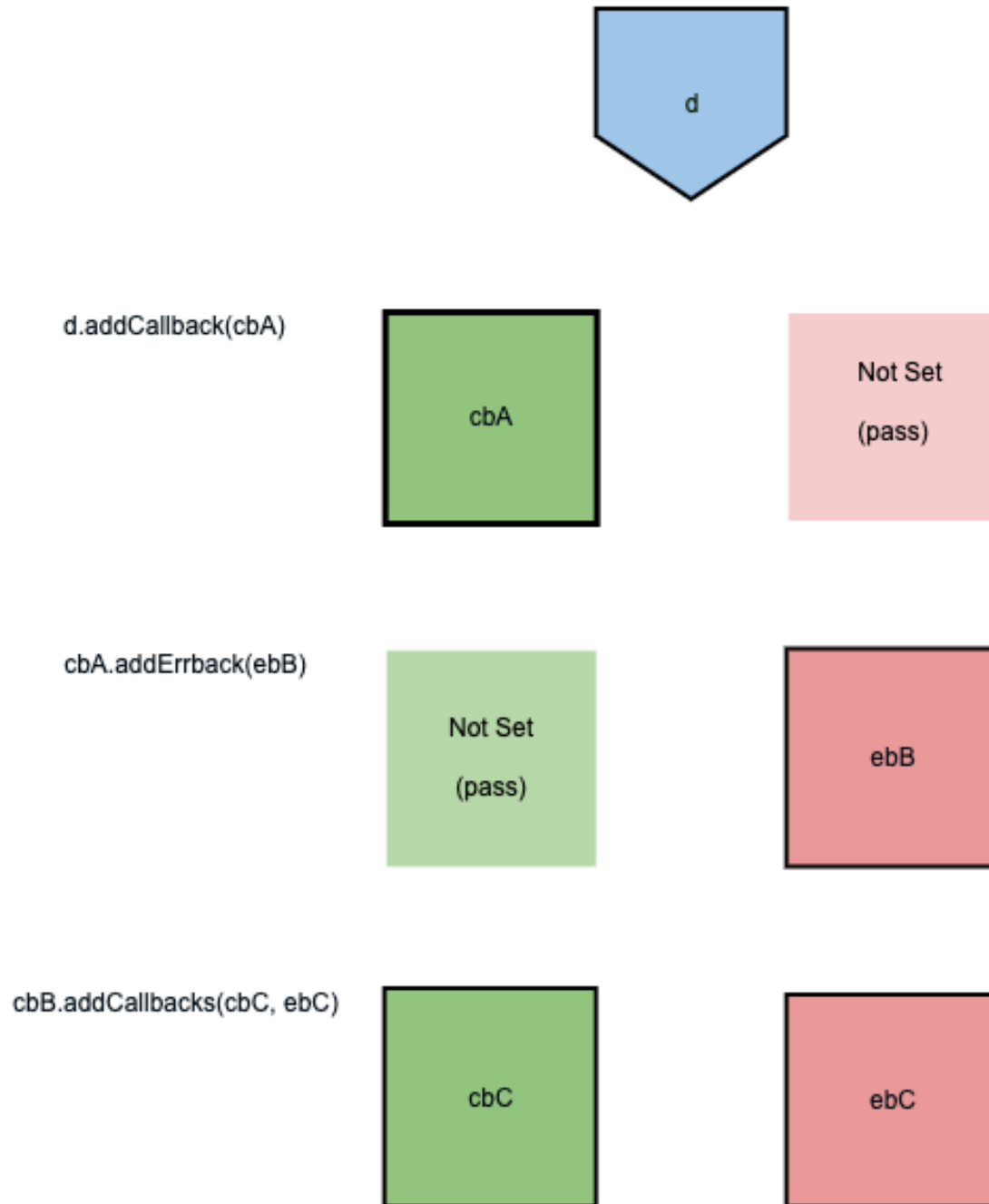
    def cbB(data):
        d = defer.Deferred()
        d.addErrback(ebC)
        d.errback(1)
        return d

    def ebC(data):
        raise Exception("NO")

    d = defer.Deferred()
    d.addCallback(cbA)
    d.callback("OK")
    return d
```

Notice that in the diagram, matching errbacks for `cbA` and `cbB`, and a matching callback for `cbC` were not set or used, but still existed. This is because callbacks and errbacks are always created in pairs, much like `try-catch` blocks. This behaviour is demonstrated below.

Callbacks and errbacks are always created in pairs



Note: Calling `addCallbacks()` *explicitly* adds both a callback **and** an errback, whereas `addCallback()`

or `addErrback()` will only *explicitly* add one **or** the other.

In this diagram, the Deferred, `d` was given a callback via the function `Deferred.addCallback`, and the function it called back (`cbA`) added an errback via the function `Deferred.addErrback`.

In both of these cases, only either a callback **or** an errback were explicitly added by the user, but in both cases a matching placeholder opposite was added by Twisted. This is because errbacks and callbacks are always created in pairs.

Notice that `cbB` adds both a callback (`cbC`) **and** an errback (`ebC`) **explicitly** via the function `Deferred.addCallbacks`.

If the `cbB` function in the first example diagram had added both a callback and errback to its Deferred as was done in this example, then the final value of `d` would be the return of whichever of `cbC` or `ebC` was eventually called (or whichever was called last).

The implementation of the above diagram would look something like this:

```
# Demonstrate that callbacks and errbacks are created in pairs
def testCallbackErrbackPairs(self):

    def cbA(data):
        d = defer.Deferred()
        d.addErrback(ebB)
        #do something that will call d.errback(x)
        return d

    def ebB(data):
        d = defer.Deferred()
        d.addCallbacks(cbC, ebC)
        #do things that will call either:
        # d.callback(x) or d.errback(x)
        return d

    def cbC(data):
        return True

    def ebC(data):
        return False

    d = defer.Deferred()
    d.addCallback(cbA)
    #do something that will eventually call d.callback(x)
    return d
```

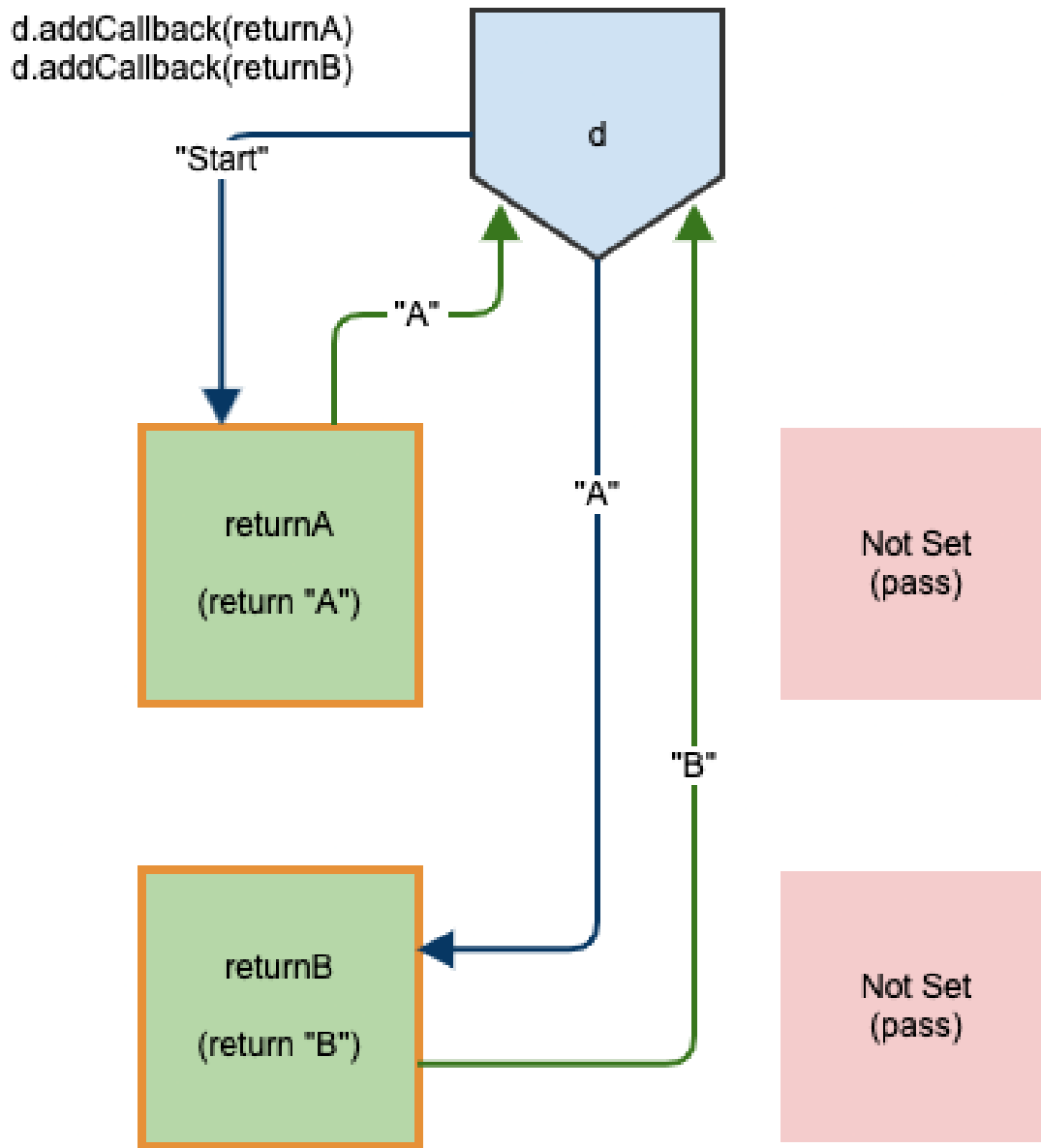
When a chain of callbacks and errbacks is useful

Callback chains are useful for handling situations where a branching tree is an ideal way to handle data. (“Run `x`. If `x` fails, do `y` unless the failure is `z`, then do...”.) They are especially suited to the task because they are modular, easy to read, and do not result in large, unwieldy walls of `if` statements.

Multiple Callbacks for a single Deferred

It is worth noting that a single Deferred can have multiple callbacks. Below, both `returnA` and `returnB` are added as callbacks to `d`. Because it has multiple callbacks, `d` is assigned the value returned by the **last** callback it fires. This

means that the eventual resolved value of `d` is “B”.



```
class ExampleTests(unittest.TestCase):

    # Demonstrate that an individual Deferred can have multiple callbacks
    def testMultipleCallbacks(self):

        def returnA(d):
            print(d)
            return "A"

        def returnB(d):
            print(d)
            return "B"
```

(continues on next page)

(continued from previous page)

```
d = defer.Deferred()
d.addCallback(returnA)
d.addCallback(returnB)
d.callback("Start")
print(d)
return d
```

When run, it should output something like this:

```
Ran 1 test in 0.125s

OK

Process finished with exit code 0
Start
A
<Deferred at 0x10b20ef98 current result: 'B'>
```

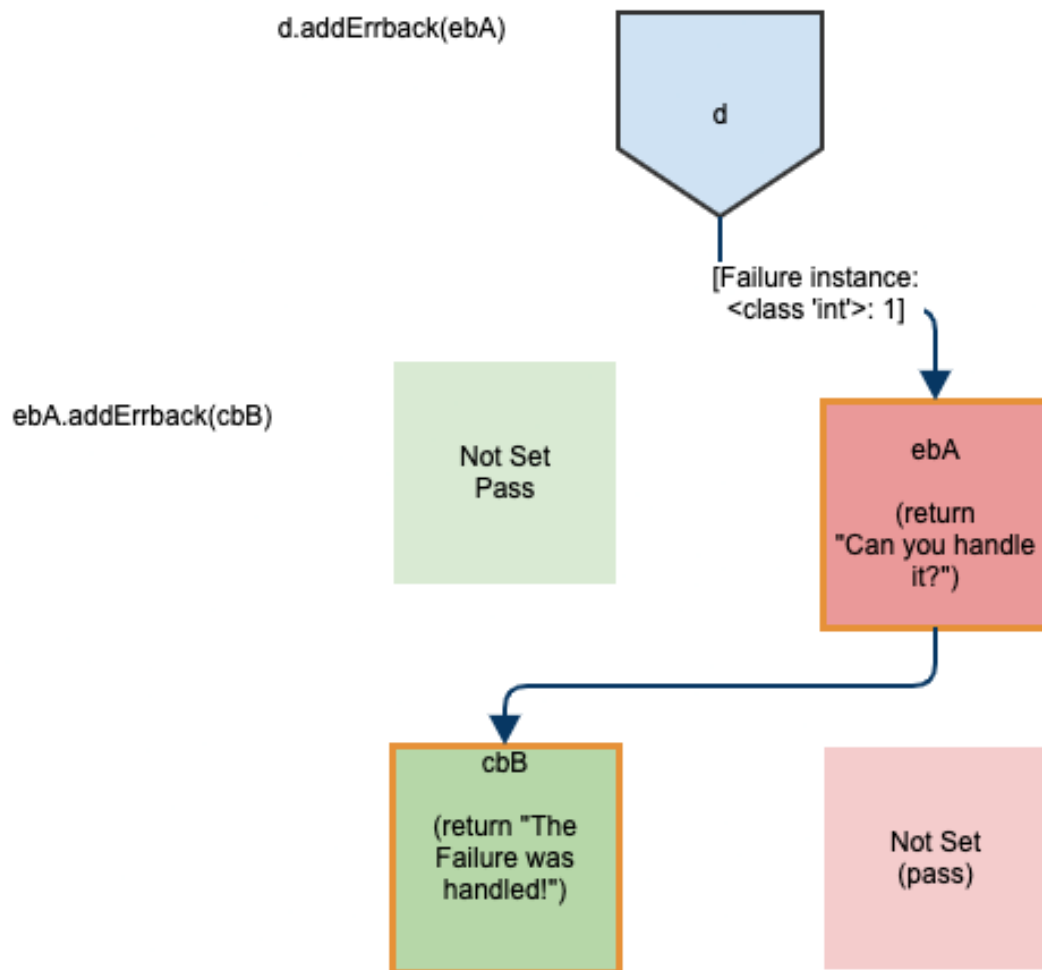
An errback calling a callback and continuing as-normal

Just because a Deferred's errback is called, it does not necessarily mean that it will continue to errback, or that it will finally resolve into a Failure instance.

Errbacks can be safely caught and called back, returning the program to normal operation. Think of errbacks as being similar to try-except blocks in standard python.

The below example creates a Deferred and has it callback `willErrback`, which does errback, but the function that it errbacks (`willCallback`) returns a callback. As a result, the errback that was created by `willErrback` is considered to be handled, allowing the program to continue.

Eventually, the original Deferred resolves with the value "It was touch and go for a bit there" because that is what the **last** callback (or errback) in the chain returns.



```
# Demonstrate an Errback continuing as a callback
def testErrbackContinues(self):

    def ebA(data):

        d = defer.Deferred()
        d.addCallback(cbB)
        d.callback("Can you handle it?")
        return d

    def cbB(data):
        print(data)
        return "The Failure was handled!"

    d = defer.Deferred()
    d.addErrback(ebA)
    d.addCallback(print)
    d.errback(1)
    return d
```

The output should look similar to this:

```
Ran 1 test in 0.113s

OK
Can you handle it?
The Failure was handled!
```

Another possible outcome could have been our errback continuing to errback any number of times before eventually being handled and calling back, returning the program to the normal flow.

Delaying Deferreds

Twisted allows users to easily schedule functions to be called after a given amount of time has passed. This functionality is especially powerful when used with **Deferreds**, to enhance their asynchronous utility.

Reactor.CallLater

The **reactor** can be used to call a function at a later date. While this returns an **IDelayedCall**, not a Deferred, it can be used to trigger a Deferred's callback.

How to use it

Below, we create a Deferred, d, and ask the reactor to:

1. Wait 1 second.
2. Call its **callback** with the argument "OK".

While **reactor.callLater** can be used with Deferreds, it could also invoke any function with any argument after any period of time:

```
from twisted.internet import reactor, task, defer
from twisted.trial import unittest

class ExampleTests(unittest.TestCase):

    def testCallLater(self):

        # Creates a Deferred and has the reactor call it back
        d = defer.Deferred()
        reactor.callLater(1, d.callback, "OK")
        return d
```

Which should output:

```
Ran 1 test in 1.006s

OK
```

task.deferLater

The **twisted.internet.task** library revolves around scheduling functions to execute at a later date, and managing these schedules. One useful function in the **task** library is **deferLater**, which calls a function after a given amount of time and returns a Deferred that fires with the eventual return value of the callable function.

How to use it

Below, we call `task.deferLater`, telling it to ask the reactor to:

1. Wait 1 second.
2. call `print` with the argument “OK”.

This returns a `Deferred` which will eventually fire with the result of the called function. Because `print` does not return a value, `d` will fire with a result of `None`:

```
def testDeferLater(self):
    # Creates a Deferred and has the reactor call it back
    d = task.deferLater(reactor, 1, print, "OK")
    assert isinstance(d, defer.Deferred), "deferLater returns a Deferred"
    return d
```

When run, the output should look like this:

```
Ran 1 test in 1.007s

OK

Process finished with exit code 0
OK
```

task.LoopingCall

The task library contains the `LoopingCall` class, which can be instantiated and told to call a given function with provided arguments. The instance of `LoopingCall` can have its `start(interval, now)` function called to begin a loop of calling the given function every interval seconds. The ‘now’ argument can be passed to it as a boolean to indicate whether it should start immediately, or wait until the interval has passed once before beginning.

`LoopingCall` is used frequently within `Peek`.

How to use it

The below code creates an instance of **`LoopingCall`**, `c`, and tells it that when it is invoked, it should call `print` with the argument “I’ll be back”.

Then, it runs `c.start`, telling it to loop every 1 second. Because `True` is not passed as a second argument, it will not call `print` immediately, but will instead wait until 1 second has passed before beginning. `c.start` returns a `Deferred` which will resolve when `c.stop` is called or when the function passed to `c` when it is created raises an error:

```
def testLoopingCall(self):
    # Allows a function to be called repeatedly
    c = task.LoopingCall(print, "I'll be back")
    d = c.start(1)
    reactor.callLater(3, c.stop)
    return d
```

When run, this should output something like:

```

Process finished with exit code 0
I'll be back
I'll be back
I'll be back
I'll be back

Ran 1 test in 3.009s

OK

```

Deferred Failures

A Deferred is an object that represents a promise that it will eventually become something else. Either it will be called back, and its eventual result will be the final value returned to its callback, or it will fail and become an instance of the `Failure` class.

The failure class helps to circumvent some of the inconveniences that standard Python error mechanisms cause for asynchronous programs. An instance of the Failure class contains frames, which provide information about the file, function, and line that an error occurred, and the Failure instance can collect multiple frames, eventually finishing its errback chain with a stack containing one or more frames, each of which describes an error.

Failure.value()

The exception instance responsible for causing a failure can be checked by calling `Failure.value()` on a Failure instance. This is useful for quickly identifying the problem in situations where an initial exception has caused cascading errors.

How to use it

Below, we create a function `assessFailValue(d)` which examines the `value` variable of the Failure to determine what to do next. We then instantiate a Deferred with an errback and callback and then we cancel it, passing it to `assessFailValue(d)`.

`assessFailValue(d)` examines the Failure instance that our Deferred has become and determines whether or not the first exception that was raised is one that should be handled further.

Because we are assuming in our example that a cancelled Deferred was cancelled by the user, and is therefore not really a bad thing, and may be something we want to treat like a callback, we can actually chain our errback into a callback with the value “Eventual success”.

This is passed back up the callback chain and ‘d’ finishes by firing its original callback and printing “Eventual Success”. If this last bit was a little confusing, don’t worry, we’re going to learn more about it in the next tutorial.

```

class ExampleTests(unittest.TestCase):

    def testFailureValue(self):

        # Set up a function to examine our Failure
        def assessFailValue(d) -> Deferred:

            # If the first exception is a CancelledError, a user cancelled it
            if type(d.value) == CancelledError:

```

(continues on next page)

(continued from previous page)

```
# So we don't consider it an error past this point
return succeed("Eventual Success!")
else:

    # /continue to handle the error elsewhere/
    return fail(1)

#create a deferred that we will fail
d = Deferred()
d.addErrback(assessFailValue)
d.addCallback(print)

# Pretend a user cancelled our Deferred, causing it to errback
d.cancel()
return d
```

Which should output something like:

```
Ran 1 test in 0.137s

OK

Process finished with exit code 0
Eventual Success!
```

Failure.check()

Failures are useful for handling errbacks. For example, if a server uses a Deferred to make a request that eventually times out, the Deferred can errback with a `TimeoutError` and a `CancelledError` and it can probably be safely ignored. However, if that same Deferred raises a `TypeError` it could be an indication that something more important has gone wrong.

A convenient way to filter out the less important errors is the `Failure.check()` function, which checks to see if a failure instance is in a predetermined list. If the failure is in the list, it returns the match, otherwise it returns `None`. This makes it easy to create a rule that would ignore a `TimeoutError` and log or notify on a `TypeError`.

How to use it

Below, much like in the first example, we check to see if the failure was raised by an exception that we should be concerned about, but this time we do it using the Failure class's inbuilt `.check()` function, which returns either `None`, or the kind of exception thrown, allowing us to easily check a list of exceptions (a number of exceptions passed to the function as an argument, not a list data structure). As in the first example, because we raised a `CancelledError`, the exception was found and returned as a callback:

```
def testFailureCheck(self):

    # Set up a function to examine our Failure
    def assessFailCheck(d) -> Deferred:
```

(continues on next page)

(continued from previous page)

```

# If the first exception is one we are ok with
if d.check(TimeoutError, CancelledError, 1) != None:

    # So we don't consider it an error past this point
    return succeed("Eventual Success!")
else:
    # /continue to handle the error elsewhere/

    return fail(1)

#create a deferred that we will fail
d = Deferred()
d.addErrback(assessFailCheck)
d.addCallback(print)

# Pretend a user cancelled our Deferred, causing it to errback
d.cancel()
return d

```

Cancelling a Deferred

Cancellation causes a Deferred to be **cancelled**, usually causing an **Errback** without the Deferred having to have naturally thrown an exception. (Note, some operations cannot be **cancelled** and the Deferred may resolve before it is **cancelled**)

When it is useful

When getting the eventual result of a Deferred is not a requirement, or could even be a hindrance. Especially if the Deferred's resolution is resource-intensive. As an example, if a user starts to connect to a distant server and the connection takes a long time, they may simply wish to exit the operation and attempt something else, maybe connecting to a different server. In that case, we do not need or want the Deferred which results from that connection anymore. We could simply ignore the result when it resolves, but it is better for us to **cancel** it.

How to use it

Calling cancel

A deferred can be cancelled simply by calling that Deferred object's `cancel()` function

As you can see below, we create a Deferred and then immediately call its `cancel()` function. Running the below code block should result in it failing with a `CancelledError`. We have added an assertion to our unit test to confirm this behaviour:

```

from twisted.internet.defer import CancelledError, Deferred
from twisted.trial import unittest

class CancellingExampleTests(unittest.TestCase):
    def testCancelsImmediately(self):
        # Create a Deferred and Cancel it def cancelImmediately() -> Deferred:
        d = Deferred()
        d.cancel()
        return d

```

```
# Returning a pre-cancelled Deferred fails with a CanceledError
self.assertFailure(cancelImmediately(), CanceledError)
```

which outputs something like:

```
Ran 1 test in 0.103s
OK
```

Custom canceller

At the time a deferred is created, a canceller can be passed to the deferred. Below, we create a Deferred, `d`, and pass a function as an argument to its `__init__(self, canceller)` function. This assigns `printCancelled` as our custom canceller. We then set an **errback**, and tell `d` to timeout in 5 seconds. 5 seconds after running it, the timeout should **cancel** `d`, which is then passed to the custom canceller:

```
def testCustomCanceller(self):

    # The Custom Canceller function
    def printWhenCancelled(d: Deferred) -> Deferred:
        print("I was Cancelled")
        return d

    # Add a custom canceller and immediately call it
    def createWithCusomCanceller() -> Deferred:
        d = Deferred(printWhenCancelled)
        d.cancel()
        return d

    self.assertFailure(createWithCusomCanceller(), CanceledError)
```

which outputs something like:

```
OK
I was Cancelled
```

Our custom canceller might have been expected to avoid calling an **errback**, as we specified which canceller would be called. This is not necessarily the case <<https://github.com/twisted/twisted/blob/twisted-20.3.0/src/twisted/internet/defer.py#L522>>‘_. If a callback is not called explicitly, it will eventually **fail** and **errback** as a **CanceledError**. We added a test case to the unit test to demonstrate this.

Running this should cause the `createWithCusomCanceller` function to be run and failed with a **CanceledError**, but first, `printWhenCancelled` should be called as the canceller and print “I was Cancelled”.

Pre-Called Deferreds

Twisted allows users to create instances of ‘pre-called’ Deferreds by calling `defer.succeed(someValue)` or `defer.fail(errorType)`, this is a convenience feature that is the same as calling:

```
d = defer.Deferred()
d.callback(someValue)
```

```
for defer.succeed() or:
```

```
d = defer.Deferred()
d.errback(errorType)
```

```
for defer.fail()
```

example

Below we have created a pre-called Deferred, and a pre-failed Deferred:

```
from twisted.internet.defer import CancelledError, fail, succeed
from twisted.trial import unittest

class ExampleTests(unittest.TestCase):

    def testPreCalledSuccess(self):

        # Create a deferred that has already resolved
        deferredPreCalledSuccess = succeed("OK")

        # Create a Deferred and Cancel it
        deferredPreCalledFail = fail(1)

        # Returning a pre-cancelled Deferred fails with a CancelledError
        self.assertEqual("OK", deferredPreCalledSuccess.result)
        self.assertFailure(deferredPreCalledFail, int)
```

Running the code should result in the test passing similarly to below:

```
Ran 1 test in 0.151s

OK
```

Our `self.assertFailure` function passing on our `preCalledFail`, would normally indicate that at some point during normal operation it had raised an exception and become a `Failure` instance.

Our `preCalledSuccess` having a result would normally indicate that it had resolved with a result of “OK”. In both cases we were able to skip past these steps and simply return a pre-fired Deferred.

Timing-Out a Deferred

Twisted’s `defer.addTimeout` allows a time limit to be set for the resolution of a Deferred. If the Deferred does not resolve in the specified time, it will be **Errback**-ed with a **CancelledError**.

When it is useful

Deferred objects are not guaranteed to resolve in any concrete amount of time, but often after a reasonable time limit is reached it is reasonable to assume that something has gone wrong and that the Deferred should be **cancelled** with an error.

As an example, consider an attempt to connect to a remote server. If the connection is taking too long to become established, it may be better to simply **cancel** the attempt and try again with a different server.

How to use it

Example

To add a Timeout to a Deferred, simply call `addTimeout()` on the Deferred object.

Below, inside of our unit test, we create a Deferred `d` and call its `addTimeout` function, which automatically **cancels** it in 5 seconds and uses the **reactor** to track the passage of time before cancellation.

Deferreds cancelled by `addTimeout` raise a `TimeoutError` AND, as they call `cancel`, a `CancelledError` as well. We have added assertions to our unit test to confirm this behaviour:

```
from twisted.internet import reactor
from twisted.internet.defer import TimeoutError, CancelledError, Deferred
from twisted.trial import unittest

class DeferredExampleTest(unittest.TestCase):

    def testTimeOutIn5(self):
        def timeOutIn5() -> Deferred:
            d = Deferred()
            d.addTimeout(5, reactor)
            d.addErrback(print)
            return d

        # A timed-out Deferred should always raise a TimeoutError
        self.assertRaises(TimeoutError, self.addCleanup(timeOutIn5))
        self.assertRaises(CancelledError, self.addCleanup(timeOutIn5))
```

Sample Output

```
Ran 1 test in 10.013s

OK
```

Using The DeferredList

The Twisted **DeferredList** is a subclass of `Deferred` that acts as aList of Deferreds. When the final Deferred in the last has fired, the **DeferredList** will call any **callbacks** that have been added to it and resolve into a 'resultList' with the format `[(bool, value)]` indicating for each Deferred whether the Deferred succeeded or failed and the final value of the Deferred.

When to use it

A **DeferredList** is useful for when you want to monitor the state of -or add the same callback to- a group of Deferreds. These are used in Peek.

How to use it

When run, the below functions should perform similarly, printing the resolved values of both the Deferreds in the **DeferredList** and the regular Deferreds, but for a large number of Deferreds, the **DeferredList** is able to save many lines and provides added functionality. Note that when used with the `@inlineCallbacks` wrapper, the entire DeferredList can be yielded, as opposed to each Deferred needing to be yielded individually.:

```
from twisted.internet.defer import DeferredList, inlineCallbacks
from twisted.internet.task import deferLater
from twisted.internet import reactor
from twisted.trial import unittest

class InlineExampleTest(unittest.TestCase):

    def testWithDeferredList(self):

        # Manage multiple Deferreds with a DeferredList
        d1 = deferLater(reactor, 1, lambda: "Called1")
        d2 = deferLater(reactor, 2, lambda: "Called2")
        dList = DeferredList([d1, d2])
        dList.addCallback(print)
        return dList

    @inlineCallbacks
    def testWithInlineDeferredList(self):

        # Manage multiple Deferreds with a DeferredList
        d1 = deferLater(reactor, 1, lambda: "Called1")
        d2 = deferLater(reactor, 2, lambda: "Called2")
        dList = DeferredList([d1, d2])
        dList.addCallback(print)
        yield dList
        return dList

    @inlineCallbacks
    def testWithoutDeferredList(self):

        # Manage multiple Deferreds without a DeferredList
        d1 = deferLater(reactor, 1, lambda: "Called1")
        d2 = deferLater(reactor, 2, lambda: "Called2")
        d1.addCallback(print)
        d2.addCallback(print)
        yield d1
        yield d2
        return d1, d2
```

testWithDeferredList should output something like:

```
Ran 1 test in 2.008s

OK

Process finished with exit code 0
[(True, 'Called1'), (True, 'Called2')]
```

testWithInlineDeferredList should output something like:

```
[(True, 'Called1'), (True, 'Called2')]
Ran 1 test in 2.009s

OK
```

testWithoutDeferredList should output something like:

```
Called1

Ran 1 test in 2.007s

OK
Called2
```

Other Useful Information: Parameters

When the DeferredList is created, it can be passed a List of Zero or more Deferreds followed by a number of parameters.

fireOnOneCallback

Fires the list's callbacks as soon as a single Deferred inside of it succeeds. This is useful if you need to know immediately whether or not any Deferred has succeeded.

fireOnOneErrback

More commonly used than **fireOnOneCallback**. Fires the list's callbacks as soon as a single Deferred inside of it is failed. This is useful if you need to know immediately whether or not any Deferred has failed.

consumeErrors

Can still be used with **fireOnOneErrback**. Prevents individual Deferreds from firing their errback chains. Instead errbacks are converted to callback results of None. This serves to prevent a large number of unhandled error messages from being logged.

Using the Deferred Semaphore

Twisted's **DeferredSemaphore** object allows the user to limit the amount of work that can be handled at once.

When to use it

You may be wondering why you would want to impose a limit on asynchronous behavior. After all, the ability to do multiple things seemingly at once is the major draw of asynchronous computing. Consider the following scenario:

Imagine an asynchronous platform needs to send out a large backlog of emails once per day. We know that as soon as Twisted's reactor starts, it will begin firing all Deferreds that are ready.

If the list of emails to send is particularly large, it could be attempting to send hundreds or thousands at once, leaving no available resources to do anything else. [This could quickly become too much for the server to handle.](#)

The **Deferred Semaphore** is perfect for situations where we need to limit the amount of work being processed while dealing with a large backlog.

[These are used in Peek.](#)

How to use it

The below code takes a list of items to process and handles their execution. To help it manage the amount of resources it dedicates to processing a potentially endless list, it creates a `DeferredSemaphore`, 'sem'.

This `DeferredSemaphore` runs the list of items to process for us and keeps track of how many items from the list are being processed at once. The `DeferredSemaphore` will execute up to `maxRun` unit of work at once, using `Deferreds` to keep track of the count, and then it will wait until one of it's `Deferreds` have fired before continuing:

```
from twisted.internet import defer
from twisted.trial import unittest

class ExampleTest(unittest.TestCase):

    def testProcessWithSemaphore(self):
        # Setup variables
        maxRun = 2
        workList = [1, 2, 3, 4, 5, 6]
        deferrals = []

        # Have a Semaphore run functions
        # and create Deferreds from them in a controlled manner
        sem = defer.DeferredSemaphore(maxRun)
        for work in workList:
            d = sem.run(print, work)
            deferrals.append(d)

        # (Use a Deferredlist to await all callbacks)
        dList = defer.DeferredList(deferrals)
        dList.addCallback(print)

        return dList
```

Which should output something like:

```
1
Ran 1 test in 0.103s
OK
2
3
4
```

(continues on next page)

(continued from previous page)

```
5
6
[(True, None), (True, None), (True, None), (True, None), (True, None), (True, None)]
```

Chaining Deferreds

The callbacks of Deferreds can be linked via the `Deferred.chainDeferred` function. This makes it so that calling or errbacking the **first** deferred also triggers the callback or errback of the second.

Functionally, this is the same as adding the callback and errback of the second deferred as a callback and errback of the first respectively.

`chainDeferred` is a one-way link. Calling back the second Deferred will not trigger the first's callback unless the second has had its own `chainDeferred` function called on the first.

When to use it

When you want one Deferred to trigger the callbacks of others.

How to use it

Below we create two deferreds: `deferredGetsChained`, and `deferredMakesChain`. Then we call `chainDeferred` on `deferredMakesChain` with `deferredGetsChained` as an argument.

Because `deferredGetsChained.callback` is now the first of `deferredMakesChain`'s callbacks, the value of `deferredMakesChain` is passed to the first callback of `deferredGetsChained`, which is `print`. Because `deferredGetsChained`'s callback has been triggered, it fires its callbacks itself, resulting in it also being printed.:

```
from twisted.internet import defer
from twisted.trial import unittest

class ExampleTests(unittest.TestCase):

    def testDeferredChaining(self):

        # Create two Deferreds
        deferredGetsChained = defer.Deferred()
        deferredMakesChain = defer.Deferred()

        # Give the chain-ee a callback
        deferredGetsChained.addCallback(print, "So was I!")

        # Chain the first Deferred to the second
        deferredMakesChain.chainDeferred(deferredGetsChained)

        # Call the first Deferred back to trigger the callback of the other
        deferredMakesChain.callback("I was called back!")
```

The output when run should look like this:


```
I was called back! So was I!
```

Potentially unexpected behavior

If a second Deferred were to be chained to the first, the first would be printed twice, although its initial value would only be printed once.

In this case, `None` will be the value of `deferredMakesChain` at the time that it is called for the second time, as its initial value has already been passed to the first chained Deferred's callback. `deferredMakesChain` is still used in both callbacks and is even processed before the values of the chained deferreds themselves:

```
from twisted.internet import defer
from twisted.trial import unittest

class ExampleTests(unittest.TestCase):

    def testDeferredChaining(self):

        # Create two Deferreds
        deferredGetsChained = defer.Deferred()
        deferredAlsoGetsChained = defer.Deferred()
        deferredMakesChain = defer.Deferred()

        # Give the chain-ee a callback
        deferredGetsChained.addCallback(print, "So was I!")
        deferredAlsoGetsChained.addCallback(print, "Me too!")

        # Chain the first Deferred to the second
        deferredMakesChain.chainDeferred(deferredGetsChained)
        deferredMakesChain.chainDeferred(deferredAlsoGetsChained)

        # Call the first Deferred back to trigger the callback of the other
        deferredMakesChain.callback("I was called back!")
```

Which results in something like:

```
Ran 1 test in 0.127s

OK
I was called back! So was I!
None Me too!
```

Common Gotchas

Deferreds can occasionally throw up some surprises by exhibiting behaviours that are not bugs, but are also likely to be different than the intended outcome the user was hoping to achieve. Because a lot of these mistakes are caught by unit tests, the examples provided in this section will not occur within a unit test, but instead within a typical reactor loop.

yield not returning anything

Inside of an `@inlineCallback` wrapper, the `yield` keyword can be used to tell the program to wait for a `Deferred` or anything that returns a `Deferred`, to resolve before continuing. If you forget to add the wrapper, however, a generator object will be returned instead.

example

In our example below, we assign the variable `username` to the return of `yieldWithoutInlineCallbacks()`, which does not return the name we were hoping for, but instead returns a generator object:

```
from twisted.internet import reactor, defer

def yieldWithoutInlineCallbacks():

    # We create a Deferred
    d = defer.Deferred()

    # Pretend to enter a name
    d.callback("Jimmy")

    # Accidentally call yield without @inlineCallbacks
    yield d
    return d

if __name__ == "__main__":

    # This function will return a generator object instead of a yielded Deferred, and
    ↪ we never noticed.
    username = yieldWithoutInlineCallbacks()

    # Some time later...
    print("Hello, ", username)
    reactor.run()
```

When run, the above code block outputs something like:

```
Hello, <generator object yieldWithoutInlineCallbacks at 0x10977e1b0>
```

Probably not what we had in mind...

Not waiting for a function that returns a Deferred

Because `Deferreds` are not generally intended to be waited for, instead using callbacks to continue whenever they are ready, at some points a user may accidentally forget to wait for a `Deferred` when the program is not intended to continue until the `Deferred` has been called. This can result in strange behavior.

example

Here we create a Deferred that is intended to get a username. We add getUsername to it as a callback, but we forget to wait for it to come back (in this example we don't call it at all). Because of this, it is assigned to the variable username, which is then printed, leading odd output:

```
from twisted.internet import reactor, defer

def returnUncalledDeferred() -> defer.Deferred:

    def getUsername():

        # The callback that is never called
        return "Jimmy"

    # We create a Deferred
    d = defer.Deferred()

    # Add a callback that will get a name
    d.addCallback(getUsername)

    # Accidentally return the Deferred before called its callback
    return d

if __name__ == "__main__":

    # This function will return a Deferred object instead of a username
    username = returnUncalledDeferred()
    print("Hello, ", username)
    reactor.run()
```

Which results in an eventual output of:

```
Hello, <Deferred at 0x10671c0f0>
```

Because the Deferred did not raise an exception, and does not list the place that the Deferred originated like in the yield example, this can be a more difficult kind of mistake to track down.

3.2.3 Twisted Challenges

Challenge 1: Identity Server and Client

- challenge1/question/identity_server.py
- challenge1/question/identity_client.py

This challenge tests your knowledge on the concepts of Transport, Protocol and ProtocolFactory in Twisted.

In this challenge, we have a server and a client that are based the Echo server/client in Twisted official documentation. Instead of sending the request back and forth between the peers, the server responds with a string of the client's IP and port (identity).

Q1: Echo Client's IP and Port as Response

- Make proper changes to implement the following:
 - Client Accepts user inputs from `stdin` and sends the inputs with `<enter>` key
 - Server responds with a message saying You are from `<ip>` on port `<port>`, saying "bla bla bla"
 - Then, server immediately loses the connection after the response
-

Q2: Display Connection History

- Based on Q1, make further changes to implement:
 - Record a history of IP and port of connected clients.
 - When client sends a message `show log`, respond a **minified** json structured as below

```
[
  {
    "ip": "1.2.3.4",
    "port": 60000
  },
  ... ,
  ... ,
  ... ,
  {
    "ip": "1.2.3.5",
    "port": 60001
  }
]
```

Sample Answer:

- `challenge1/answer/identity_server.py`
- `challenge1/answer/identity_client.py`

Challenge 2: Deferred 101

- `challenge2/question/fetch_webpage.py`
-

This challenge tests the basics of `Deferred` objects.

In this challenge, we have a simple synchronous HTTP client that needs refactoring. Your challenge is to change this code with Twisted to implement the features below.

Q1: Fetch the webpage

- Read function `synchronousCall`
 - Implement a function called `chainedCall` as an asynchronous version with `Deferred`
-

- Pass the test case `testChainedCall`
-

Q2: Wrap your call chains with `inlineCallbacks`

- Based on the previous question, wrap the function `chainedCall` with `inlineCallbacks`
 - Pass the test case `testInlineCallback`
-

Sample Answer:

- `challenge2/answer/fetch_webpage.py`

Challenge 3: Message Relay

- `challenge3/question/message_relay.py`
-

This challenge tests your knowledge on chained calls with `callback` and `errBack`.

In this challenge, we have 3 persons relaying a message. Each person can either `pass(callback)` or `blame(errback)` to the next person. The behaviour of each person's `pass/blame` are pre-defined as functions in code.

Q1: Add code in function `q1` below to stop Person 3 blaming

Q2: Make the message pass through Person 2 without Person 1 blaming

Sample Answer

- `challenge3/answer/message_relay.py`

Challenge 4: Heartbeat

- `challenge4/question/heartbeat.py`
-

This challenge tests your knowledge on `loopingCall`.

The existing code is to send out heartbeat signal of "*" every 3 seconds to each session after connection. Note: this is not a *broadcast* which sends the "*" to all clients at the same time.

To test your heartbeat server, use `netcat` or `telnet`, connect to `127.0.0.1` to port `5000`.

Sample Answer

- `challenge4/answer/heartbeat.py`

Challenge 5: Who Got “Yes” First

- `challenge5/question/who_got_it_first.py`
-

This challenge tests your knowledge on `DeferredList`, `Failure` and your comprehensive applied knowledge of Twisted.

There is a web JSON API return randomly-picked answer of either yes or no. The challenge code is a client that sends 3 requests to the API.

Q1: Add code to function `getAll` to retrieve all responses and then print the collected results with `pprint` (which is provided)

Q2: Change code to function `_filterAnswer` and function `_handleUnexpectedAnswer` to get the first answer “yes” and stop waiting for all other requests. Do not interrupt if there is no answer “yes” in any response in which case `(None, 0)` will be printed.

When you are making your change, please make sure the following is satisfied:

- `_filterAnswer` filters out “yes” to the callback chain while it raises an `UnexpectedAnswerError` to error-back chain if it’s a “No”.
 - `_handleUnexpectedAnswer` handles all failures from `_filerAnswer`. It catches and stops the complaint due to `UnexpectedAnswerError` and puts the `Deferred` object back to callback chain. But it still raises any other types of failures.
-

Q3: Add code to function `_displayUnexpectedAnswer` to print the filtered-out response(s) of “No” using `print`.

Sample Answer

- `challenge5/answer/who_got_it_first.py`

Challenge 6: Concurrency Limit

- `challenge6/question/concurrency_limit.py`
-

This challenge tests your knowledge on `DeferredSemaphore`.

There is a script that gets the file sizes of gcc releases. Please add code function `download` to limit concurrent requests sent by `Deferred` objects by `maxRun`.

Sample Answer

- `challenge6/answer/concurrency_limit.py`

Challenge 7: Cancel on Timeout

- `challenge7/question/cancel_on_timeout.py`
-

This challenge tests your knowledge on cancellation and timeout.

The existing code needs a line of code to set a timeout of 2 seconds before cancel the deferred if timeout is reached. On timeout, please use `logTimeout` when it is timed out.

Sample Answer

- `challenge7/answer/cancel_on_timeout.py`

Challenge 8: Chain Them Up

- `challenge8/question/original.py`
-

This challenge tests your knowledge on chaining deferreds.

The original code implements a callback chain. Please refactor the two lines of `addCallbacks` as an oneliner with `chainDeferred`.

Sample Answer

- `challenge8/answer/refactored.py`

Challenge 9: Work with Synchronous Functions

- `challenge9/question/original.py`
-

This challenge tests your knowledge on integrating synchronous functions.

The original code implements a `Deferred` object from a function. please reactor the function to return a `Deferred`.

Sample Answer

- `challenge9/answer/refactored.py`

3.2.4 Understanding VortexPy

Introducing VortexPy

What is a VortexPy?

VortexPy is Synerty's observable, routable, data serialisation and transport code.

Understanding DeferUtil

deferUtil

DeferUtil provides utility methods make it easy to integrate Twisted's Deferreds simply, with a focus on painless multithreading. The library also provides control methods to limit or log the execution of these methods.

deferToThreadWrapWithLogger

Explanation

This method is a decorator used to send blocking methods to threads easily.‘‘ This can only be done from the main thread inside of the reactor event loop.

Returns

Deferred

Usage

```
import logging
logger = logging.getLogger(__name__)
@deferToThreadWrapWithLogger(logger)
def myFunction(arg1, kw=True):
    pass
```

Example

This example did not take place in a unit test, as functions that are not used inside of Twisted's main reactor loop cannot be deferred to a separate thread.

As you can see by the output, the method `usuallyBlocking` below is moved to another thread by `@deferToThreadWrapWithLogger` removing the blockage and allowing the reactor to continue.

```
from vortex.DeferUtil import deferToThreadWrapWithLogger
from twisted.internet import reactor
import time, logging

logger = logging.getLogger(__name__)

@deferToThreadWrapWithLogger(logger)
def usuallyBlocking():
    time.sleep(2)
    print("Blocker ran outside of main thread")

if __name__ == "__main__":
    reactor.callLater(1, usuallyBlocking)
```

(continues on next page)

(continued from previous page)

```

reactor.callLater(1.1, print, "Runs in Main Thread")
reactor.callLater(3.2, reactor.stop)
reactor.run()

```

Output

The example should produce output similar to this:

```

Runs in Main Thread
Blocker ran outside of main thread

```

maybeDeferredWrap

Explanation

This is a decorator that ensures a function will return a Deferred. It allows the use of Twisted's maybeDeferred as a decorator.

Twisted's maybeDeferred calls the given function with the given arguments and returns either a Failure wrapped in `defer.fail` if the function returns a Failure object, or if an Exception is raised. Otherwise, the function's return is wrapped in `defer.succeed` and returned.

Returns

Deferred

Usage

```

@maybeDeferredWrap
someFunction():

```

Example

```

from twisted.trial import unittest
from twisted.internet.defer import Deferred
from vortex.DeferUtil import maybeDeferredWrap

class ExampleTest(unittest.TestCase):

    def testVortex(self):

        @maybeDeferredWrap
        def notUsuallyDeferred():
            return True

        d = notUsuallyDeferred()
        self.assertIsInstance(d, Deferred)

```

The example should produce output similar to this:

```
Ran 1 test in 0.100s
OK
```

ensureDeferredWrap

Explanation

This is a decorator that ensures an async function will return a Deferred. It allows the use of Twisted’s “ensureDeferred” function as a decorator.

This keeps async processing compatible with Twisted’s reactive design while providing support for the await/async syntax (and async processing).

Returns

Deferred

Usage

```
@ensureDeferredWrap
someFunction():
```

Example

```
from twisted.trial import unittest
from twisted.internet.defer import Deferred
from vortex.DeferUtil import ensureDeferredWrap
import time

class ExampleTest(unittest.TestCase):

    def testVortex(self):

        async def wait1():
            time.sleep(1)
            return True

        @ensureDeferredWrap
        async def usuallyAsync():
            return await wait1()

        d = usuallyAsync()
        self.assertIsInstance(d, Deferred)
```

The example should produce output similar to this:

```
Ran 1 test in 1.149s
```

```
OK
```

nonConcurrentMethod

Explanation

This decorator ensures that method will only run once at any given time. It will ignore subsequent calls if the method is still running. The returned decorated method has a “running” flag which indicates whether or not it is still running.

Returns

The now-decorated argument method.

Usage

```
@nonConcurrentMethod
someFunction():
    pass
```

Example

```
from vortex.DeferUtil import nonConcurrentMethod, deferToThreadWrapWithLogger
from twisted.internet import reactor
import time, logging

logger = logging.getLogger(__name__)

@deferToThreadWrapWithLogger(logger)
@nonConcurrentMethod
def usuallyConcurrent():
    time.sleep(2)
    print("Function Called")

if __name__ == "__main__":

    reactor.callLater(0.1, usuallyConcurrent)
    reactor.callLater(0.1, usuallyConcurrent)
    reactor.callLater(3.5, reactor.stop)
    reactor.run()
```

The example should only run `usuallyConcurrent` once and will produce this output:

```
Function Called
```

```
Process finished with exit code 0
```

vortexLogFailure

Explanation

Checks to see if the `Failure` object passed to it as `failure` has already been assigned the attribute `_vortexLogged`. If it has not, then the `Logger` passed to it as `loggerArg` will be called and will emit a log event.

Finally, if `consumeError` is `True`, then it will return `SuccessValue`, otherwise, `failure` will be returned.

Returns

Either `successValue` if successful or `failure` if not.

Usage

```
vortexLogFailure(failure, loggerArg, consumeError=False, successValue=True)
```

`vortexLogFailure` is used to easily log a `Failure` that had not already been logged by `vortex`.

Example

```
from twisted.trial import unittest
from twisted.internet import defer
from vortex.DeferUtil import vortexLogFailure
import logging

logger = logging.getLogger(__name__)

class ExampleTest(unittest.TestCase):

    def testVortex(self):

        def alwaysFails():
            d = defer.Deferred()
            d.addErrback(vortexLogFailure, logger, consumeError = True)
            d.errback(1)
            alwaysFails()
```

The example should produce output similar to this:

```
Ran 1 test in 0.103s

OK
Traceback (most recent call last):
Failure: builtins.int: 1
```

Thread Checks

`VortexPy` allows the user to easily check which thread things are running in and ensure that the correct thread is being used.

isMainThread

Explanation

Returns `True` if the caller is in the main thread, otherwise returns `False`

Returns

`bool`

Usage

```
isMainThread()
```

noMainThread

Explanation

Ensure the caller **IS NOT** running in the main thread, raise an Exception if it is.

Returns

`None`

Usage

```
noMainThread()
```

yesMainThread

Explanation

Ensure the caller **IS** running in the main thread, raise an Exception if it is not.

Returns

`None`

Usage

```
yesMainThread()
```

Example

Code Sample

```
from vortex.DeferUtil import deferToThreadWrapWithLogger, isMainThread, noMainThread, \
    yesMainThread
from twisted.internet import reactor
import logging

logger = logging.getLogger(__name__)

@deferToThreadWrapWithLogger(logger)
def notInMainThread():

    # isMainThread returns a bool indicating whether it is in the main thread
    print("notInMainThread running in main thread:", isMainThread())

    # noMainThread does not return a bool,
    # it raises an exception if it is in the main thread.
    noMainThread()

def inMainThread():

    # isMainThread returns a bool indicating whether it is in the main thread
    print("inMainThread running in main thread:", isMainThread())

    # yesMainThread does not return a bool,
    # it raises an exception if it is not in the main thread.
    yesMainThread()

if __name__ == "__main__":

    reactor.callLater(0.1, notInMainThread)
    reactor.callLater(0.1, inMainThread)
    reactor.callLater(3.2, reactor.stop)
    reactor.run()
```

Output

```
inMainThread running in main thread: True
notInMainThread running in main thread: False
```

3.3 Python Gotchas

Python has a few key differences when compared to other, compiled languages, which occasionally cause unexpected behaviour.

3.3.1 List Iteration Modification

Description

```
nums[1, 2, 3, 4, 3, 2, 1]
print("Before:", nums)

for ind, n in enumerate(nums):
    if n < 3:
        del(nums[ind])

print("After:", nums)
```

Expected Result

```
Expected: nums [3, 4, 3]
```

Actual Output

```
After: [2, 3, 4, 3, 1]
```

Explanation

`del` modifies in-place. When index 0 (value: 1) in `nums` is deleted, index 1 (value: 2) is moved down to index 0 to take its place BEFORE the iterator increments. This allows the value 2 to be essentially skipped-over, despite being less than 3.

Instead, use list comprehension to create a new list that meets the same requirements.

3.3.2 Mutable Class Variables

Description

```
class C:
    var = []
    val = 0

    def __init__(self, val):
        self.var.append(val)
        self.val = val

x = C(1)
y = C(2)
```

What would be the output from printing the values of `x` and `y`?

Expected Result

```
y.var: [1]
y.val: 1

y.var: [1, 2]
y.val: 2
```

OR

```
y.var: [1,2]
y.val: 2

y.var: [1, 2]
y.val: 2
```

Actual Output

```
y.var: [1, 2]
y.val: 1

y.var: [1, 2]
y.val: 2
```

Why this causes confusion

Users who have become accustomed to other programming languages may expect that Python's class instances would not share values unless those values were static. If this were the case, `x.val` would also be 2, as that is what `y` overwrote it to be.

What is happening

Python default class variables behave similarly to static variables in other languages, every individual instance of a class references the same class variable. However, if a default value is overridden it becomes local to the instance instead of overwriting the class variable.

Because there is a matching variable inside of the class, there is no need to keep looking at a wider scope for it and so the class variable is never checked.

Because when it was initialised, `x` modified `var`, but overwrote `val`, it still sees the changes `y` made to `var`, but is not affected by further changes to `val`, as these would exist in the class scope and not the instance scope.

Demonstration

```
class C:
    var = []

    def __init__(self, x):
        self.var.append(x)

    def overrideVar(x):
        self.var = [x]
```

(continues on next page)

(continued from previous page)

```
x = C(1)
y = C(2)
y.overrideVar(x)
z = C(4)

print("x:", x.var)
print("y:", y.var)
print("z:", z.var)
```

Will output:

```
x: [1, 2, 4]
y: [3]
z: [1, 2, 4]
```

Proving (because z can still access it) that the class variable remains unchanged, but that the variable in y's instance scope has been overridden.

3.3.3 Mutable Default Function Arguments

Description

```
def func(val, arg1=[], arg2={}):
    arg1.append(val)
    arg2[val] = str(val)
    print("arg1:", arg1)
    print("arg2:", arg2)
func(1)
func(2)
func('c')
```

What does the function print on the call to `func('c')`?

Expected Result

```
arg1: ['c']
arg2: {'c': 'c'}
```

Actual Result

```
arg1: [1, 2, 'c']
arg2: {1: '1', 2: '2', 'c': 'c'}
```

Explanation

Default function arguments in Python are persistent. The function is created once and then called multiple times. The default arguments are mutable, so despite being persistent they can be overwritten.

If your intention is to have the default values created fresh each time, then one option would be to give the default argument a value of `None` and then use an if check to set it to what is wanted:

```
def func(val, arg1=None, arg2=None):  
  
    # Set the arguments to the desired defaults here  
    if arg1 is None:  
        arg1 = []  
    if arg2 is None:  
        arg2 = {}  
  
    arg1.append(val)  
    arg2[val] = str(val)
```

This would give the original expected output of:

```
arg1: ['c']  
arg2: {'c': 'c'}
```

3.4 ReStructuredText Cheat Sheet

The following tips should help you get started writing reStructuredText

3.4.1 Sections

Sections are created by underlining (and optionally overlining) the section title with a punctuation character, at least as long as the text and a blank line before and after.

These section titles and headings will be used to create the contents when the documentation is built.

Note: The Page Title can be seen at the top of this page, *Add Documentation*.

Header 1 can be seen at the top of this section, *Sections*.

3.4.2 Header 2

Sample paragraph.

Header 3

Sample paragraph.

If you expand the page contents you will notice that “Header 3” isn’t available in the page contents. This is because the maxdepth of the toctree is ‘2’. see *TOC tree*

This is an example of the “Add Documentation”(Page Title), “Sections”(Header 1), “Header 2”, and “Header 3” raw text:

```

=====
Add Documentation
=====

Sections
-----

Header 2
\ \ \ \ \ \ \ \

Header 3
\ \ \ \ \ \ \ \

```

Instruction Divider

Four dashes with a leading blank line and following blank line.

```

-----

```

Text Formatting

The following roles don't do anything special except formatting the text in a different style.

3.4.3 Inline Markups

Inline markup is quite simple, some examples:

- one asterisk: `*text*`, *text* for emphasis (italics),
- two asterisks: `**text**`, **text** for strong emphasis (boldface), and
- backquotes: `:code:`text``, `text` for code samples.

3.4.4 Files

The name of a file or directory. Within the contents, you can use curly braces to indicate a “variable” part, for example:

```
learn_plugin_development/LearnPluginDevelopment_AddDocs.rst
```

```
:file:`learn_plugin_development/LearnPluginDevelopment_AddDocs.rst`
```

3.4.5 Reference Links

Reference link names must be unique throughout the entire documentation.

Place a label directly before a section title.

The link name will match the section title.

Add Documentation

An example of the reference link above the section title:

```
.. _learn_plugin_development_add_docs:

=====
Add Documentation
=====
```

An example of the reference link:

```
:ref:`learn_plugin_development_add_docs`
```

3.4.6 URL Link

A raw link can be entered without a title, but if a title is entered be sure to leave a space before the URL address:

[Synerty](http://www.synerty.com/)

```
`Synerty <http://www.synerty.com/>`_
```

3.4.7 Code Block

Two semi-colons followed by a blank line and two leading tabs for each line of code. The code block is ended by contents written without leading tabs.

```
this.code
```

```
::

    this.code
```

3.4.8 Bullets

- First point
- Second point

```
- First point
- Second point
```

3.4.9 Numbered Lists

1. First point
2. Second point

```
#. First point
#. Second point
```

Directives are indicated by an explicit markup start ‘.. ‘ followed by the directive type, two colons, and whitespace (together called the “directive marker”). Directive types are case-insensitive single words.

3.4.10 Images

The filename given must either be relative to the source file, or absolute which means that they are relative to the top source directory.



```
.. image:: peek_plugin_tutorial/synerty_logo_400x800.png
```

3.4.11 Admonitions

Admonitions are specially marked “topics” that can appear anywhere an ordinary body element can. They contain arbitrary body elements. Typically, an admonition is rendered as an offset block in a document, sometimes outlined or shaded, with a title matching the admonition type.

Note: Multi Line NOTE

Mutli Parapgraph

- Can contain bullets
- 1. numbers points

and references: *Add Documentation*

```
.. note:: Multi
    Line
    NOTE

    Mutli Parapgraph

    - Can contain bullets

    #. numbers points

    and references: :ref:`learn_plugin_development_add_docs`
```

3.4.12 TOC tree

This directive inserts a table of contents at the current location, including sub-TOC trees.

Document titles in the toctree will be automatically read from the title of the referenced document.

Here is an example:

```
=====
Example Documentation
=====

.. toctree::
    :maxdepth: 2
    :caption: Contents:

    intro
    strings
    datatypes
    numeric
    (many more documents listed here)
```

3.4.13 Docstring Format

This extension `sphinx.ext.autodoc`, can import the modules you are documenting, and pull in documentation from docstrings in a semi-automatic way.

Warning: `autodoc` imports the modules to be documented. If any modules have side effects on import, these will be executed by `autodoc` when `sphinx-build` is run. If you document scripts (as opposed to library modules), make sure their main routine is protected by a `if __name__ == '__main__':` condition.

A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition.

All modules should normally have docstrings, and all functions and classes exported by a module should also have docstrings. Public methods (including the `__init__` constructor) should also have docstrings. A package may be documented in the module docstring of the `__init__.py` file in the package directory.

Example:

```
"""
This is a reST style.

:param param1: this is a first param
:param param2: this is a second param
:returns: this is a description of what is returned
:raises KeyError: raises an exception
"""
```

Below is an abstract from file `peek_plugin_tutorial/_private/server/LogicEntryHook.py`, create in the step *Add File LogicEntryHook.py*.

```
def load(self) -> None:
    """ Start

    This will be called to start the plugin.
    Start, means what ever we choose to do here. This includes:

    - Create Controllers

    - Create payload, observable and tuple action handlers.
```

(continues on next page)

(continued from previous page)

```
"""  
logger.debug("Loaded")
```

Below is an abstract from file peek-plugin-base/peek_plugin_base/PeekPlatformCommonHookABC.py

```
class PeekPlatformCommonHookABC(metaclass=ABCMeta):  
  
    @abstractmethod  
    def getOtherPluginApi(self, pluginName:str) -> Optional[object]:  
        """ Get Other Plugin Api  
  
        Asks the plugin for its api object and return it to this plugin.  
        The API returned matches the platform service.  
  
        :param pluginName: The name of the plugin to retrieve the API for  
        :return: An instance of the other plugins API for this Peek Platform  
                Service.  
  
        """
```

Setup OS Requirements

4.1 Setup OS Requirements Windows

The Peek platform is designed to run on Linux, however, it is compatible with windows. Please read through all of the documentation before commencing the installation procedure.

TODO: Upgrade to PostgreSQL 12 TODO: Add timescale support

4.1.1 Installation Objective

This *Installation Guide* contains specific Windows operating system requirements for the configuring of synerty-peek.

Required Software

Some of the software to be installed requires internet access. For offline installation some steps are required to be installed on another online server for the files to be packaged and transferred to the offline server.

Below is a list of all the required software:

- Microsoft .NET Framework 3.5 Service Pack 1
- Visual C++ Build Tools 2015
- PostgreSQL 10.4+
- Node.js 7+ and NPM 5+
- Python 3.6
- Virtualenv
- FreeTDS
- Msys Git

Optional Software

- 7zip
- Notepad ++
- Installing Oracle Libraries (Instructions in the procedure)

Installation of 7zip is optional. This tool will come in handy during the process but is not required.

Installation of Notepad ++ is optional. Notepad ++ is a handy tool for viewing documents and has useful features.

Installing Oracle Libraries is required if you intend on installing the peek agent. Instruction for installing the Oracle Libraries are in the *Online Installation Guide*.

4.1.2 OS Commands

The config file for each service in the peek platform describes the location of the BASH interpreter. Peek is coded to use the bash interpreter and basic posix compliant utilities for all OS commands.

When peek generates it's config it should automatically choose the right interpreter.

```
"C:\Program Files\Git\bin\bash.exe" if isWindows else "/bin/bash"
```

4.1.3 Installation Guide

The following sections begin the installation procedure.

4.1.4 Create Peek OS User

Create a windows user account for peek with admin rights. Search for **Computer Management** from the start menu, and create the new peek user from there.

Warning: Make sure the username is all lower case.

Account Type Administrator

Username peek

Password PA\$\$WORD

Sign in to the peek account.

Important: All steps after this point assume you're logged in as the peek user.

Tip: Run the “**control userpasswords2**” command from the run window to have peek automatically login. This is useful for development virtual machines.

4.1.5 MS .NET Framework 3.5 SP1

Online Installation:

Download <http://download.microsoft.com/download/2/0/e/20e90413-712f-438c-988e-fdaa79a8ac3d/dotnetfx35.exe>

From <https://www.microsoft.com/en-ca/download>

Offline Installation:

Download <https://download.microsoft.com/download/2/0/E/20E90413-712F-438C-988E-FDAA79A8AC3D/dotnetfx35.exe>

Note: Restart if prompted to restart.

4.1.6 Visual C++ Build Tools 2015

Online Installation:

Download <http://go.microsoft.com/fwlink/?LinkId=691126>

From <http://landinghub.visualstudio.com/visual-cpp-build-tools>

Offline Installation:

Install using the ISO

Download <https://www.microsoft.com/en-US/download/details.aspx?id=48146>

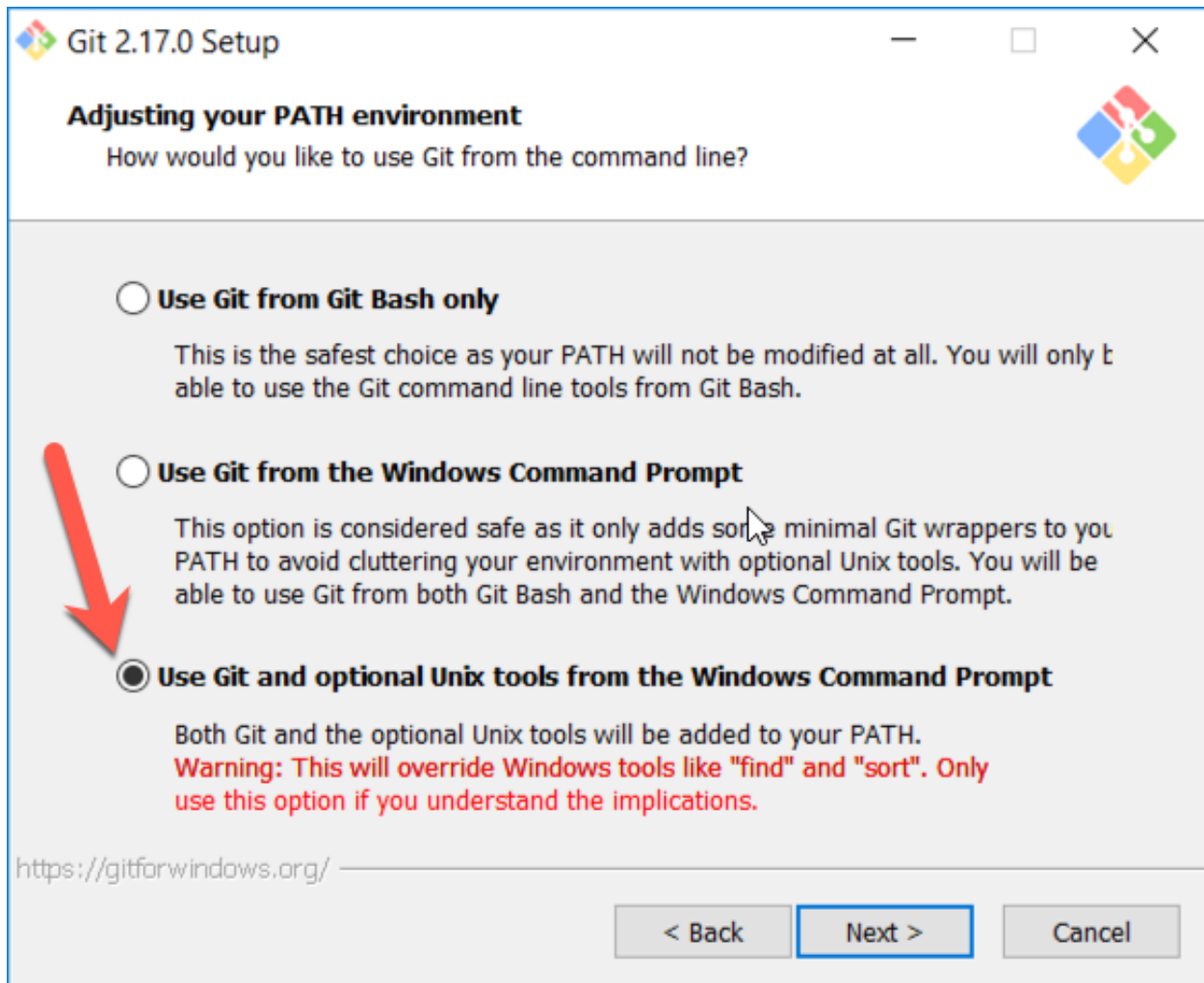
4.1.7 Setup Msys Git

Download <https://github.com/git-for-windows/git/releases/download/v2.17.0.windows.1/Git-2.17.0-64-bit.exe>

From <https://git-for-windows.github.io>

Use all default options, Except on the **Adjusting your PATH environment** screen.

On the “Adjusting your PATH environment” screen, select “Use Git and optional Unix tools from the Windows Command Prompt”



Note: This is equivalent to adding “C:\Program Files\Git\mingw64\bin” and “C:\Program Files\Git\usr\bin” to the system PATH environment variable.

Open a new command window, and type **bash**, it should find the bash command.

Press Ctrl+D to exit bash.

Open a new command or powershell window, and type **git**, it should find the git command.

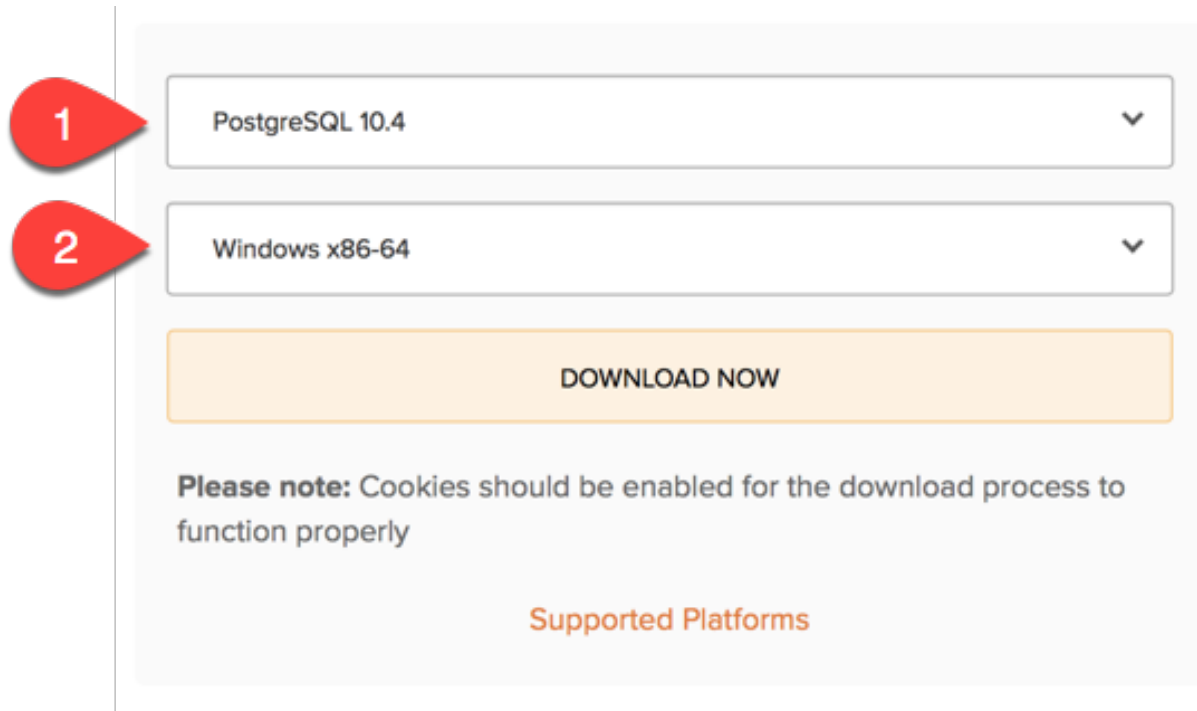
4.1.8 Install PostgreSQL

Peek requires PostgreSQL as it's persistent, relational data store.

Download <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads#windows>

From <https://www.postgresql.org>

Note: Ensure you download the 64bit version or PostgreSQL or the Peek windows service dependencies will not recognise it (“postgresql-10” vs “postgresql-x64-10”)



1 PostgreSQL 10.4

2 Windows x86-64

DOWNLOAD NOW

Please note: Cookies should be enabled for the download process to function properly

[Supported Platforms](#)

Install PostgreSQL with default settings.

Make a note of the postgres user password that you supply, you'll need this.

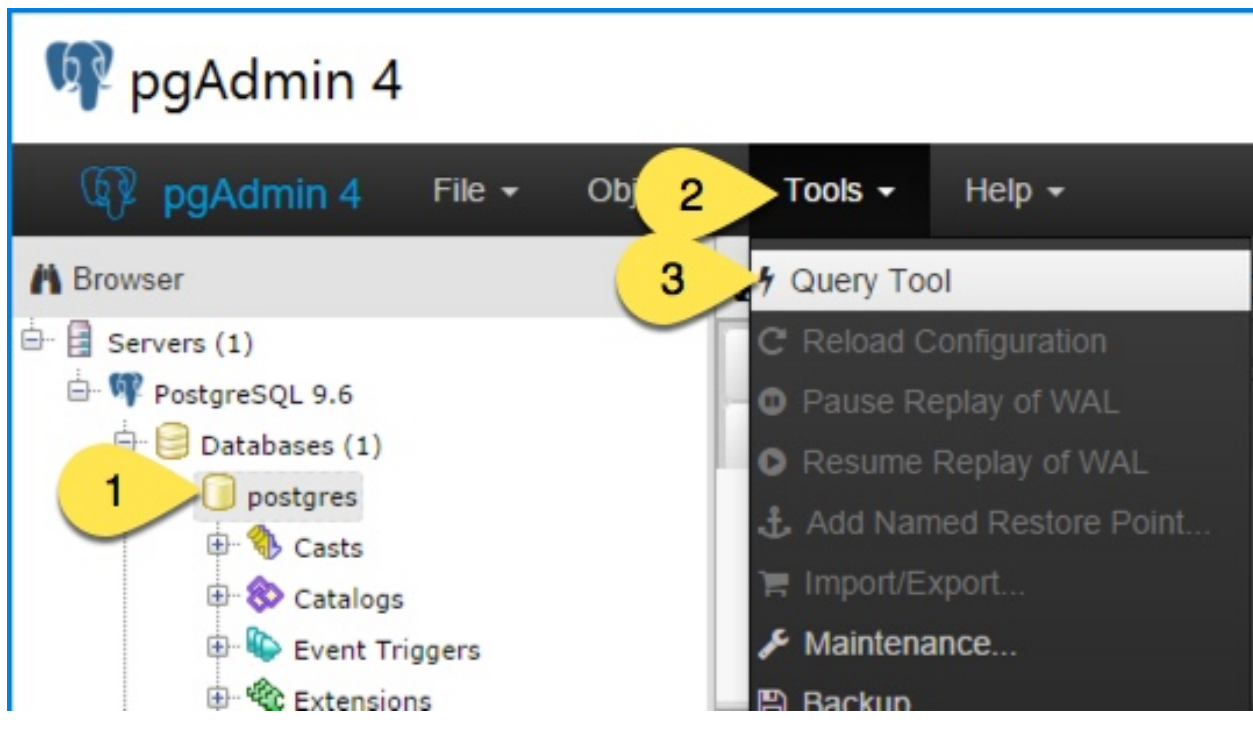
Warning: Generate a strong password for both peek and postgres users for production use.

Synerty recommends 32 to 40 chars of capitals, lower case and numbers, with some punctuation, best to avoid these ‘ / \ ‘ “

<https://strongpasswordgenerator.com>

Run pgAdmin4

Open the Query Tool

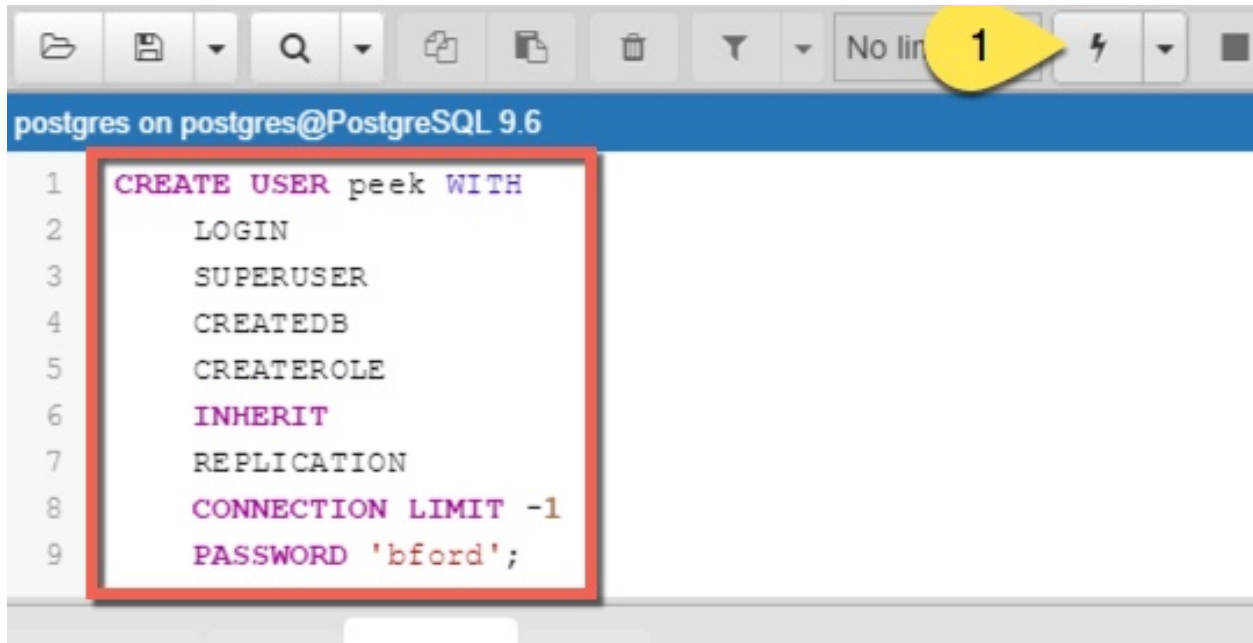


Create the peek user, run the following script:

```
CREATE USER peek WITH
    LOGIN
    CREATEDB
    INHERIT
    REPLICATION
    CONNECTION LIMIT -1
    PASSWORD 'PASSWORD';
```

Note: Replace `PASSWORD` with a secure password from <https://xkpasswd.net/s/> for production.

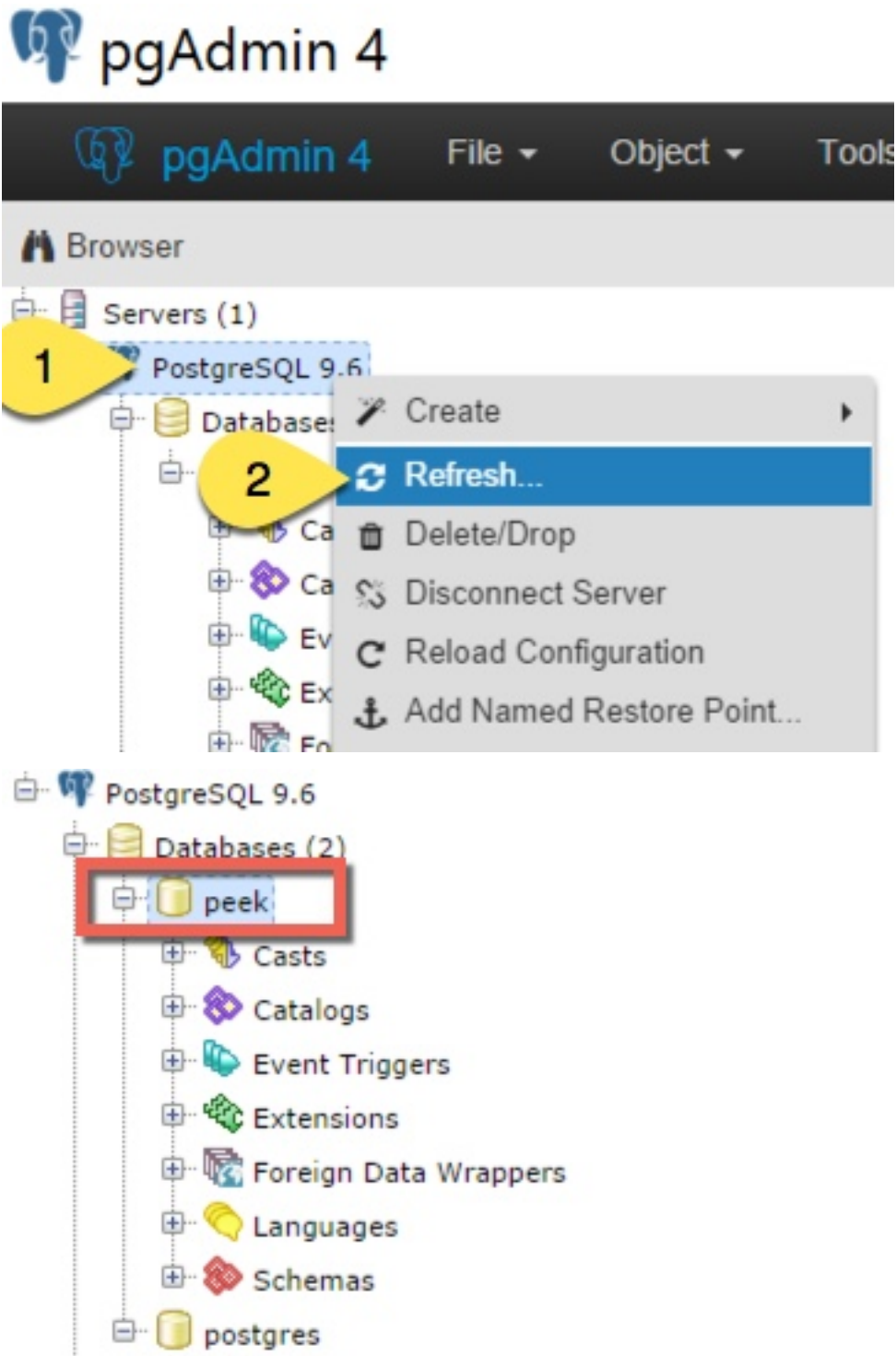
Example:



Create the peek database, run the following script:

```
CREATE DATABASE peek WITH
    OWNER = peek
    ENCODING = 'UTF8'
    CONNECTION LIMIT = -1;
```

Confirm database was created

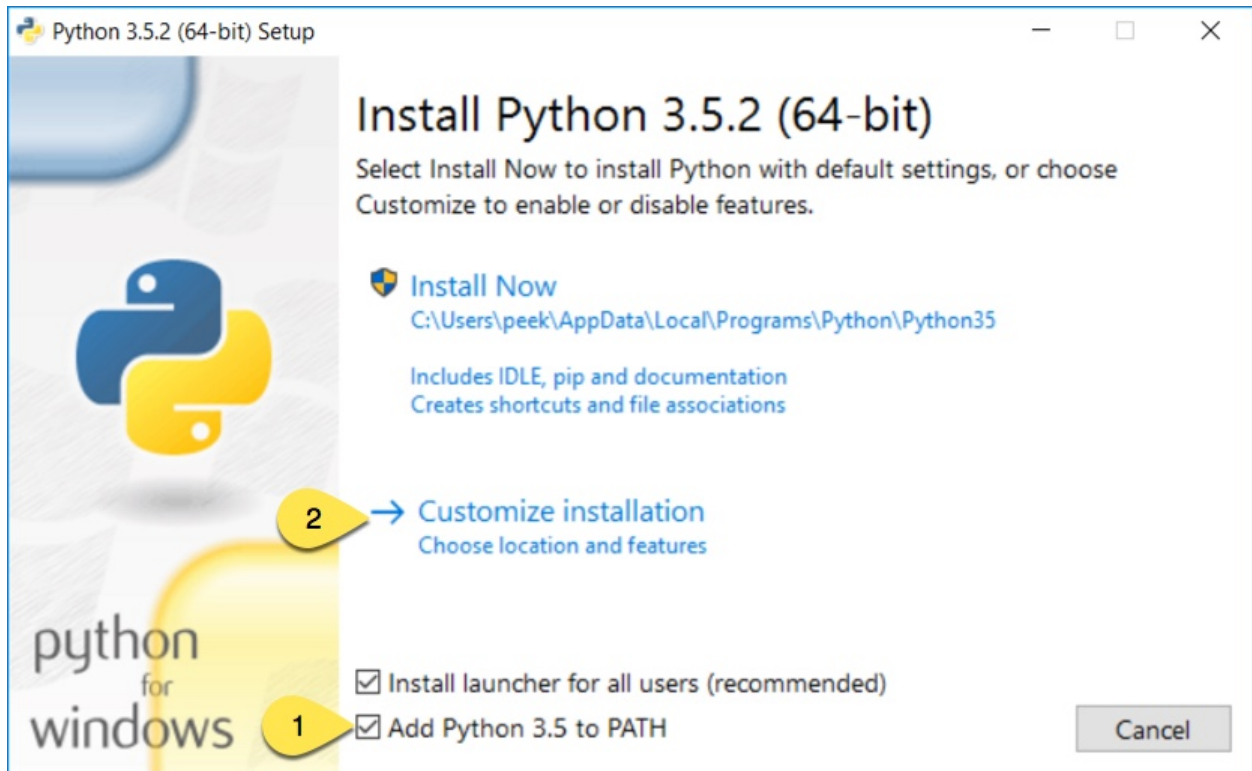


4.1.9 Install Python 3.6

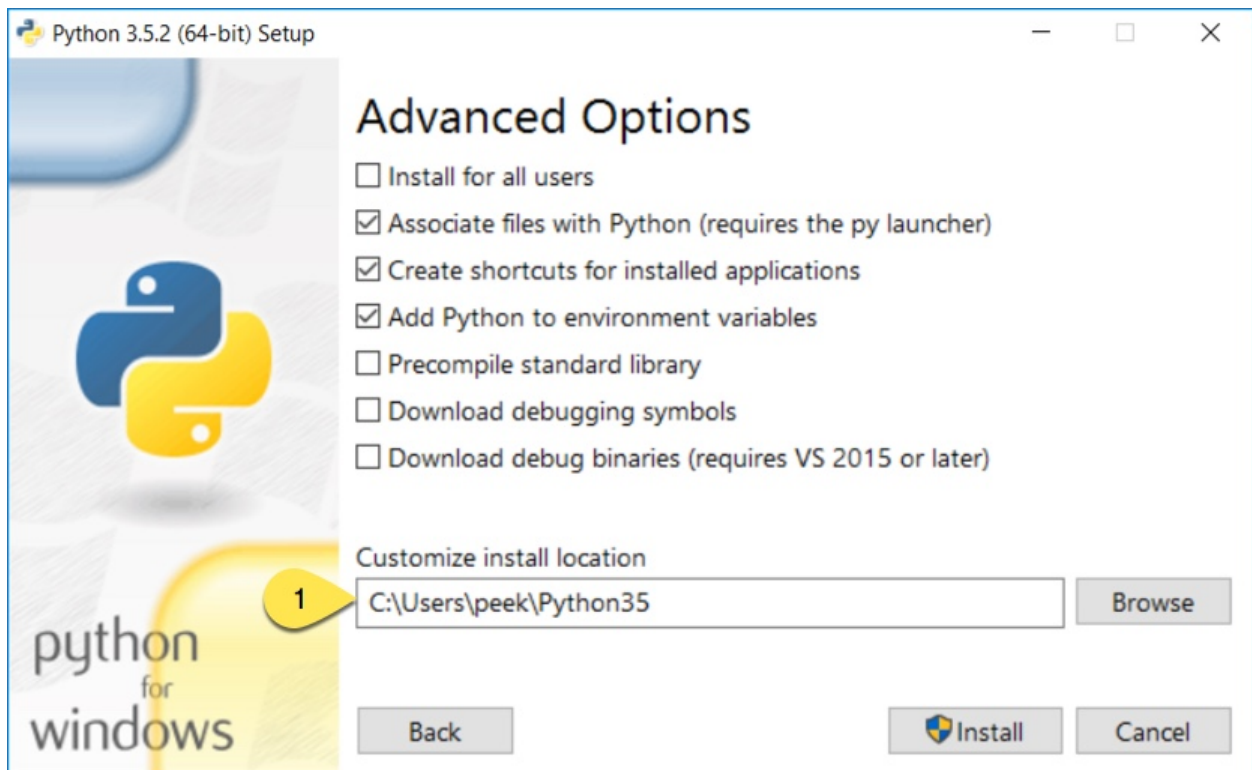
Download <https://www.python.org/ftp/python/3.9.1/python-3.9.1-amd64.exe>

From <https://www.python.org/downloads/windows/>

Check the 'Add Python 3.6 to PATH' and select 'Customize Installation'



Update the 'Customize install location' to PATH C:\Users\peek\Python36\



Confirm PATH(s) to environment variables

```
echo %PATH%  
  
...  
C:\Users\peek\Python36\  
C:\Users\peek\Python36\Scripts\  

```

Virtual Environment

synerty-peek is deployed into python virtual environments. Install the virtualenv python package

Upgrade pip:

```
pip install --upgrade pip
```

Open the command prompt and run the following command:

```
pip install virtualenv
```

The Wheel package is required for building platform and plugin releases

```
pip install wheel
```

4.1.10 Install Worker Dependencies

Install the parallel processing queue we use for the peek-worker-service tasks.

Download and install Redis:

Download <https://github.com/MicrosoftArchive/redis/releases/download/win-3.0.504/Redis-x64-3.0.504.msi>

Download and install Erlang:

Download http://erlang.org/download/otp_win64_20.0.exe

Download and install RabbitMQ:

Download https://github.com/rabbitmq/rabbitmq-server/releases/download/rabbitmq_v3_6_10/rabbitmq-server-3.6.10.exe

Under Control Panel -> System -> Advanced system settings

Add the following to PATH in the “System” environment variables

```
C:\Program Files\RabbitMQ Server\rabbitmq_server-3.6.10\sbin
```

Tip: On Win 10, enter “environment” in the task bar search and select **Edit the system environment variables**

Enable the RabbitMQ management plugins:

```
rabbitmq-plugins enable rabbitmq_mqtt
rabbitmq-plugins enable rabbitmq_management
```

Confirm the RabbitMQ Management Console and the RabbitMQ MQTT Adaptor are listed under the running applications:

```
rabbitmqctl status
```

4.1.11 Install Oracle Client (Optional)

The oracle libraries are optional. Install them where the agent runs if you are going to interface with an oracle database.

Download the following from oracle.

The version used in these instructions is **18.5.0.0.0**.

1. Download the ZIP “Basic Package” instantclient-basic-windows.x64-18.5.0.0.0dbru.zip from <http://www.oracle.com/technetwork/topics/winx64soft-089540.html>
 2. Download the ZIP “SDK Package” instantclient-sdk-windows.x64-18.5.0.0.0dbru.zip from <http://www.oracle.com/technetwork/topics/winx64soft-089540.html>
-

Extract both the zip files to C:\Users\peek\oracle

Under Control Panel -> System -> Advanced system settings

Add the following to **PATH** in the “User” environment variables

```
C:\Users\peek\oracle\instantclient_18_5
```

Tip: On Win 10, enter “environment” in the task bar search and select **Edit the system environment variables**

The Oracle instant client needs msvcr120.dll to run.

Download and install the x64 version from the following microsoft site.

<https://www.microsoft.com/en-ca/download/details.aspx?id=40784>

Reboot windows, or logout and login to ensure the PATH updates.

4.1.12 Install FreeTDS (Optional)

FreeTDS is an open source driver for the TDS protocol, this is the protocol used to talk to a MSSQL SQLServer database.

Peek needs this installed if it uses the pymssql python database driver, which depends on FreeTDS.

Download https://github.com/ramiro/freetds/releases/download/v0.95.95/freetds-v0.95.95-win-x86_64-vs2015.zip

From <https://github.com/ramiro/freetds/releases>

Unzip contents into

```
C:\Users\peek
```

Rename C:\users\peek\freetds-v0.95.95 to C:\users\peek\freetds

Under Control Panel -> System -> Advanced system settings

Add the following to PATH in the “System” environment variables

```
C:\Users\peek\freetds\bin
```

Tip: On Win 10, enter “environment” in the task bar search and select **Edit the system environment variables**

Create file `freetds.conf` in C:\

```
[global]
port = 1433
instance = peek
tds version = 7.4
```

If you want to get more debug information, add the dump file line to the [global] section Keep in mind that the dump file takes a lot of space.

```
[global]
port = 1433
instance = peek
tds version = 7.4
dump file = c:\\users\\peek\\freetds.log
```

dll files

Download http://indy.fulgan.com/SSL/openssl-1.0.2j-x64_86-win64.zip

From <http://indy.fulgan.com/SSL/>

Ensure these files are in the system32 folder:

- libeay32.dll
 - ssleay32.dll
-

You will need to duplicate the above files and name them as per below:

- libeay32MD.dll
- ssleay32MD.dll

4.1.13 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

4.2 Setup OS Requirements macOS

This section describes how to perform the setup for macOS (previously OSX).

Please read through all of the documentation before commencing the installation procedure.

4.2.1 Installation Objective

This Installation Guide contains specific Mac 11.2.1 Big Sur operating system requirements for the configuring of synerty-peek.

Required Software

Some of the software to be installed requires internet access. For offline installation some steps are required to be installed on another online server for the files to be packaged and transferred to the offline server.

Below is a list of all the required software:

- Xcode (from the app store)
- Homebrew
- Python 3.6.x
- Postgres 12.x

Optional Software

- Oracle Client

Installing Oracle Libraries is required if you intend on installing the peek agent. Instruction for installing the Oracle Libraries are in the Online Installation Guide.

- FreeTDS

FreeTDS is an open source driver for the TDS protocol, this is the protocol used to talk to the MSSQL SQLServer database.

4.2.2 Installation Guide

Follow the remaining section in this document to prepare your macOS operating system to run the Peek Platform.

The instructions on this page don't install the peek platform, that's done later.

4.2.3 Set Terminal shell to Bash

Press Command + Space to bring up preferences.

Type "terminal"

Run the following command and enter the password when prompted

code:

```
chsh -s /bin/bash
```

This will change your chosen shell from zsh to Bash.

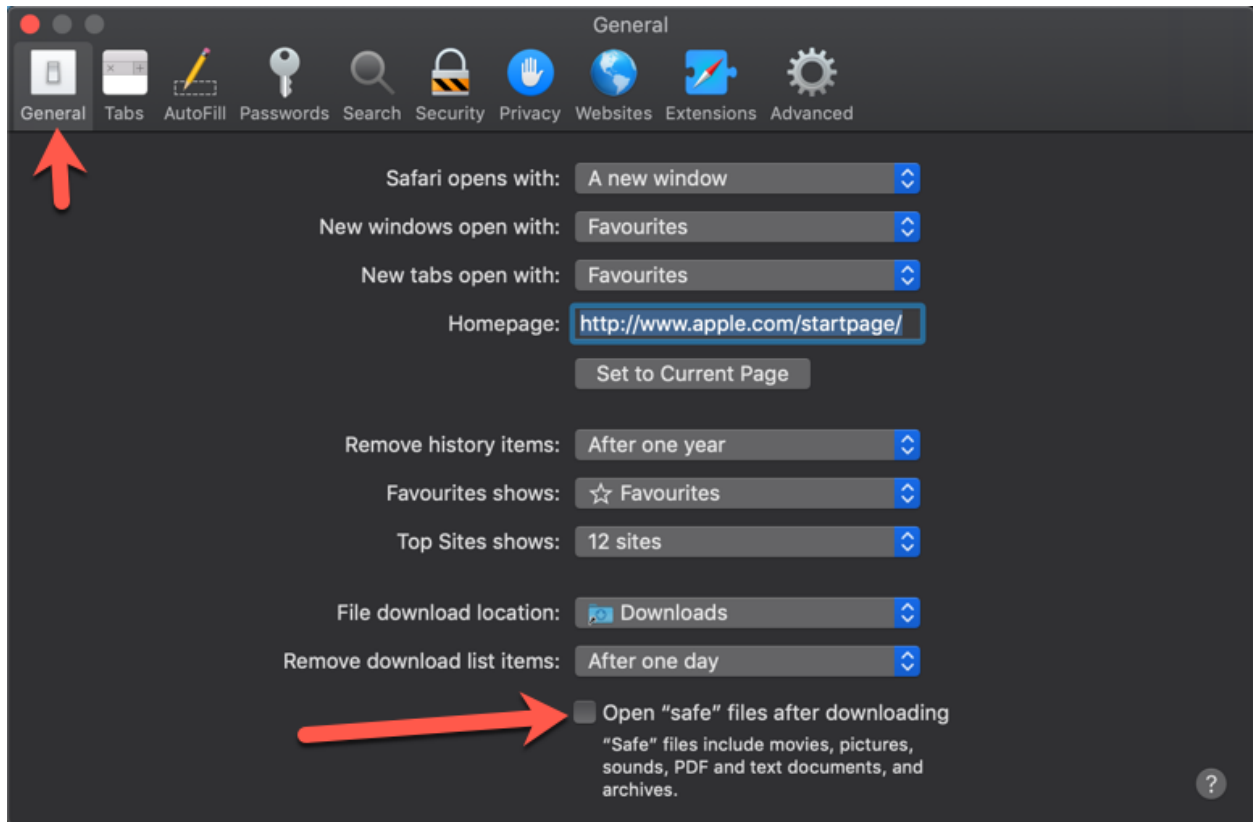
4.2.4 Safari Open Safe Files

If you're using safari on the mac (which you probably are), make sure **Open "Safe" files after downloading** is turned off.

This will cause Safari to unzip files that have been downloaded, and invalidate some of the install steps.

In Safari, press Command + , to bring up preferences.

Uncheck the **Open "Safe" files after downloading** checkbox.



Close safari preferences.

4.2.5 Create Peek Platform OS User

Alternatively to creating a peek user, if you are developing with peek you might want to Symlink the `/Users/*developerAccount*` to `/Users/peek`. If doing this run: `sudo ln -s /Users/*developerAccount*/ /Users/peek` then skip to the next step *Install Xcode*.

Create a user account for peek with admin rights.

```
sudo ln -s /Users/*developerAccount*/ /Users/peek
```

Account Type Administrator

Username peek

Password PA\$WORD

Sign in to the peek account.

Important: All steps after this point assume you're logged in as the peek user.

4.2.6 Install Xcode

From the app store, install Xcode.

Run Xcode and accept 'Agree' to the license. Xcode will then install components.

Exit Xcode

Run Terminal

Apple's Command Line Developer Tools can be installed on recent OS versions by running this command in the Terminal:

```
sudo xcode-select -r
```

A popup will appear, select 'Install' then 'Agree' to the license.

Agree to the Xcode license in Terminal run:

```
sudo xcodebuild -license
```

Type q, type agree and hit 'Enter'

4.2.7 Install Homebrew

Edit ~/.bash_profile and insert the following:

```
#### USE THE GNU TOOLS ####  
# Set PATH to gnu tools  
export PATH="`echo ~/bin:$PATH` "
```

To install Homebrew, run the following command in terminal:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/  
↪master/install)"
```

Install gnu-sed for the build scripts

```
brew install gnu-sed
```

Install wget, needed for python download


```
brew install wget
```

Create the symlinks to prefer the GNU tools

```
mkdir ~/bin
ln -s `which gsed` ~/bin/sed
```

Install the dev libs that the python packages will need to compile

```
brew install openssl@1.1 zlib openldap freetds
```

Install libgeos

```
wget https://raw.githubusercontent.com/Homebrew/homebrew-core/master/Formula/geos.rb
brew install geos.rb
rm geos.rb
```

4.2.8 Preparing .bash_profile

Edit ~/.bash_profile and insert the following:

```
##### SET THE PEEK ENVIRONMENT #####
# Setup the variables for PYTHON
export PEEK_PY_VER="3.9.1"
export PATH="/Users/peek/opt/bin:$PATH"

# Set the variables for the platform release
# These are updated by the deploy script
export PEEK_ENV=""
export PATH="${PEEK_ENV}/bin:$PATH"
```

Warning: Restart your terminal you get the new environment.

4.2.9 Install Python 3.9.1

Download and unarchive the supported version of Python

```
cd ~
source .bash_profile
wget https://github.com/python/cpython/archive/v${PEEK_PY_VER}.zip
unzip v${PEEK_PY_VER}.zip
cd cpython-${PEEK_PY_VER}
```

Configure the build

```
export LDFLAGS="-L/usr/local/opt/openssl/lib -L/usr/local/opt/zlib/lib"
export CPPFLAGS="-I/usr/local/opt/openssl/include -I/usr/local/opt/zlib/include"
export PKG_CONFIG_PATH="/usr/local/opt/openssl/lib/pkgconfig:/usr/local/opt/zlib/lib/
↳pkgconfig"

./configure \
  --prefix=$HOME/opt \
  --enable-optimizations \
  --enable-shared \
  --with-openssl=$(brew --prefix openssl)
```

Make and Make install the software

```
make install
```

Cleanup the download and build dir

```
cd
rm -rf cpython-${PEEK_PY_VER}
rm v${PEEK_PY_VER}.zip
```

Symlink the python3 commands so they are the only ones picked up by path.

```
cd /Users/peek/opt/bin
ln -s pip3 pip
ln -s python3 python
cd
```

Open a new terminal and test that the setup is working

```
pass="/Users/peek/opt/bin/python"
[ "`which python`" == "$pass" ] && echo "Success" || echo "FAILED"

pass="Python ${PEEK_PY_VER}"
[ "`python --version`" == "$pass" ] && echo "Success" || echo "FAILED"

pass="/Users/peek/opt/bin/pip"
[ "`which pip`" == "$pass" ] && echo "Success" || echo "FAILED"

pass="pip 20.2.3 from /Users/peek/opt/lib/python3.9/site-packages/pip (python 3.9)"
[ "`pip --version`" == "$pass" ] && echo "Success" || echo "FAILED"
```

Upgrade pip:

```
pip install --upgrade pip
```

The following packages are required to package/deploy the macOS release.

Note: This is required for the pymysql setup.py

```
pip install Cython
```

synerty-peek is deployed into python virtual environments. Install the virtualenv python package

```
pip install virtualenv
```

The Wheel package is required for building platform and plugin releases

```
pip install wheel
```

4.2.10 Install PostgreSQL

Install the relational database Peek stores its data in. This database is PostgreSQL 12.

Note: Run the commands in this step as the peek user.

Download the PostgreSQL source code

```
PEEK_PG_VER=12.5
SRC_DIR="$HOME/postgresql-${PEEK_PG_VER}"

# Remove the src dir and install file
rm -rf ${SRC_DIR} || true
cd $HOME

wget https://ftp.postgresql.org/pub/source/v${PEEK_PG_VER}/postgresql-${PEEK_PG_VER}.tar.bz2
tar xjf postgresql-${PEEK_PG_VER}.tar.bz2

cd ${SRC_DIR}
```

Configure and build PostGresQL

```
export CPPFLAGS="-I$(brew --prefix openssl@1.1)/include "
export LDFLAGS="-L$(brew --prefix openssl@1.1)/lib "

./configure \
    --disable-debug \
    --prefix=$HOME/opt \
    --enable-thread-safety \
    --with-openssl \
    --with-python
```

(continues on next page)

(continued from previous page)

```
make -j4

make install-world

# this is required for timescale to compile
cp ${SRC_DIR}/src/test/isolation/pg_isolation_regress ~/opt/bin
```

Remove install files to clean up the home directory

```
# Remove the src dir and install file
cd
rm -rf ${SRC_DIR}*
```

Initialise a PostgreSQL database

```
#Refresh .bash_profile so initdb can find postgres
source .bash_profile

initdb --pgdata=$HOME/pgdata/12 --auth-local=trust --auth-host=md5
```

Tune the postgresql.conf

```
F="$HOME/pgdata/12/postgresql.conf"

sed -i 's/max_connections = 100/max_connections = 200/g' $F
```

Install CMake

Download CMake source code:

```
cd $HOME
PEEK_CMAKE_VER=3.19.2
SRC_DIR="$HOME/CMake-${PEEK_CMAKE_VER}"
wget https://github.com/Kitware/CMake/archive/v${PEEK_CMAKE_VER}.zip

unzip v${PEEK_CMAKE_VER}.zip
cd ${SRC_DIR}
```

Compile CMake from source:

```
./configure --prefix=$HOME/opt

make -j6 install

# Remove the src dir and install file
cd
rm -rf ${SRC_DIR}*
rm v${PEEK_CMAKE_VER}.zip
```

Install PostgreSQL Timescaledb

Next install timescaledb, this provides support for storing large amounts of historical data.

www.timescale.com

Download the timescaledb source code

```
PEEK_TSDB_VER=1.7.4

cd
wget https://github.com/timescale/timescaledb/archive/${PEEK_TSDB_VER}.zip
unzip ${PEEK_TSDB_VER}.zip
cd timescaledb-${PEEK_TSDB_VER}
```

Install the packages:

```
export CPPFLAGS=`pg_config --cppflags`
export LDFLAGS=`pg_config --ldflags`
export OPENSSL_ROOT_DIR="/usr/local/opt/openssl/"

# Bootstrap the build system
./bootstrap -DAPACHE_ONLY=1

# To build the extension
cd build && make

# To install
make install

# Cleanup the source code
cd
rm -rf ${PEEK_TSDB_VER}.zip
rm -rf timescaledb-${PEEK_TSDB_VER}
```

Install timescaledb-tune:

```
wget https://timescalerelases.blob.core.windows.net/homebrew/timescaledb-tools-0.8.1.
↪tar.lzma
tar xf timescaledb-tools-0.8.1.tar.lzma
cp timescaledb-tools/timescaledb-tune $HOME/opt/bin
rm -rf timescaledb-tools
rm timescaledb-tools-0.8.1.tar.lzma
```

Tune the database:

```
PGVER=12
FILE="$HOME/pgdata/${PGVER}/postgresql.conf"
timescaledb-tune -quiet -yes -conf-path ${FILE} -pg-version ${PGVER}
```

Finish PostgreSQL Setup

Make PostgreSQL a service :

Note: This will require sudo permissions

Run the following command

```
# Make sure there is a LaunchAgents folder
mkdir -p ~/Library/LaunchAgents/

# Create the plist file
cat <<EOF > ~/Library/LaunchAgents/postgresql.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>KeepAlive</key>
  <true/>
  <key>Label</key>
  <string>postgresql</string>
  <key>ProgramArguments</key>
  <array>
    <string>${HOME}/opt/bin/postgres</string>
    <string>-D</string>
    <string>${HOME}/pgdata/12</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>WorkingDirectory</key>
  <string>${HOME}/opt</string>
  <key>StandardOutPath</key>
  <string>${HOME}/pgdata/12/postgres.log</string>
  <key>StandardErrorPath</key>
  <string>${HOME}/pgdata/12/postgres.log</string>
</dict>
</plist>
EOF

# Tell launchctl to use the plist
launchctl load ~/Library/LaunchAgents/postgresql.plist
```

Check that the service has started:

```
pgrep -laf postgres
```

Finish configuring and starting PostgreSQL.

Allow the peek OS user to login to the database as user peek with no password

```
F=/usr/local/var/postgres/pg_hba.conf
cat | sudo tee $F <<EOF
```

(continues on next page)

(continued from previous page)

#	TYPE	DATABASE	USER	ADDRESS	METHOD
	local	all	postgres		peer
	local	all	peek		trust
# "local" is for Unix domain socket connections only					
	local	all	all		peer
# IPv4 local connections:					
	host	all	all	127.0.0.1/32	md5
# IPv6 local connections:					
	host	all	all	:::1/128	md5
EOF					

Create the database

```
createdb -O peek peek
```

Note: If you already have a database, you may now need to upgrade the timescale extension.

```
psql peek <<EOF
ALTER EXTENSION timescaledb UPDATE;
EOF
```

Set the PostgreSQL peek users password

```
psql -d postgres -U peek <<EOF
\password
\q
EOF

# Set the password as "PASSWORD" for development machines
# Set it to a secure password from https://xkpasswd.net/s/ for production
```

Cleanup traces of the password

```
[ ! -e ~/.psql_history ] || rm ~/.psql_history
```

Finally, Download pgAdmin4 - A graphically PostgreSQL database administration tool.

Download the latest version of pgAdmin4 for macOS from the following link

<https://www.pgadmin.org/download/pgadmin-4-macos/>

4.2.11 Install Worker Dependencies

Install the parallel processing queue we use for the peek-worker-service tasks.

Redis

Install Redis via Homebrew with the following command:

```
brew install redis
```

Start redis and create a start at login launchd service:

```
brew services start redis
```

Open new terminal and test that Redis setup is working

```
pass="/usr/local/bin/redis-server"
[ "`which redis-server`" == "$pass" ] && echo "Success" || echo "FAILED"
```

Increase the size of the redis client queue

```
BEFORE="client-output-buffer-limit pubsub 64mb 16mb 90"
AFTER="client-output-buffer-limit pubsub 32mb 8mb 60"
sed -i "s/{BEFORE}/{AFTER}/g" /usr/local/etc/redis.conf

brew services restart redis
```

RabbitMQ

Install RabbitMQ via Homebrew with the following command:

```
brew install rabbitmq
```

Start rabbitmq and create a start at login launchd service:

```
brew services start rabbitmq
```

Edit ~/.bash_profile and insert the following:

```
##### SET THE RabbitMQ ENVIRONMENT #####
# Set PATH to include RabbitMQ
export PATH="/usr/local/sbin:$PATH"
```

Refresh the bash_profile:

```
source $HOME/.bash_profile
```

Open new terminal and test that RabbitMQ setup is working


```
pass="/usr/local/sbin/rabbitmq-server"
[ "`which rabbitmq-server`" == "$pass" ] && echo "Success" || echo "FAILED"
```

Enable the RabbitMQ management plugins:

```
rabbitmq-plugins enable rabbitmq_mqtt
rabbitmq-plugins enable rabbitmq_management
```

4.2.12 Install Oracle Client (Optional)

The oracle libraries are optional. Install them where the agent runs if you are going to interface with an oracle database.

Make the directory where the oracle client will live

```
mkdir ~/oracle
```

Download the following from [oracle](https://download.oracle.com/otn_software/mac/instantclient/).

The version used in these instructions is 19.8.0.0.0.

1. Download the “Basic Package” from https://download.oracle.com/otn_software/mac/instantclient/instantclient-basic-macos.zip
2. Download the “SDK Package” from https://download.oracle.com/otn_software/mac/instantclient/instantclient-sdk-macos.zip

Copy these files to ~/oracle on the peek server.

Extract the files.

```
cd ~/oracle
unzip instantclient-basic-macos.zip
unzip instantclient-sdk-macos.zip
```

Add links to \$HOME/lib to enable applications to find the libraries:

```
mkdir ~/lib
ln -s ~/oracle/instantclient_19_8/libclntsh.dylib ~/lib/
```

Edit ~/.bash_profile and insert the following:

```
##### SET THE ORACLE ENVIRONMENT #####
# Set PATH to include oracle
export ORACLE_HOME=`echo ~/oracle/instantclient_19_8`
export PATH="$ORACLE_HOME:$PATH"

##### SET THE DYLD_LIBRARY_PATH #####
export DYLD_LIBRARY_PATH="$DYLD_LIBRARY_PATH:$ORACLE_HOME"
```

4.2.13 Change Open File Limit on macOS

macOS has a low limit on the maximum number of open files. This becomes an issue when running node applications.

Make sure the sudo password timer is reset

```
sudo echo "Sudo is done, lets go"
```

Run the following commands in terminal:

```
echo kern.maxfiles=65536 | sudo tee -a /etc/sysctl.conf
echo kern.maxfilesperproc=65536 | sudo tee -a /etc/sysctl.conf
sudo sysctl -w kern.maxfiles=65536
sudo sysctl -w kern.maxfilesperproc=65536
```

Edit ~/.bash_profile and insert the following:

```
##### Open File Limit #####
ulimit -n 65536 65536
```

Restart the terminal

4.2.14 What Next?

Refer back to the [How to Use Peek Documentation](#) guide to see which document to follow next.

4.3 Setup OS Requirements RHEL

This section describes how to perform the setup for Red Hat Linux Server 7.7. The Peek platform is designed to run on Linux.

Please read through all of the documentation before commencing the installation procedure.

4.3.1 Installation Objective

This Installation Guide contains specific Red Hat Linux Server 7.4 operating system requirements for the configuring of synerty-peek.

Required Software

Some of the software to be installed requires internet access. For offline installation some steps are required to be installed on another online server for the files to be packaged and transferred to the offline server.

Below is a list of all the required software:

- Python 3.9.x
- Postgres 12.x

Suggested Software

The following utilities are often useful.

- rsync
- git
- unzip

Optional Software

- Oracle Client

Installing Oracle Libraries is required if you intend on installing the peek agent. Instruction for installing the Oracle Libraries are in the Online Installation Guide.

- FreeTDS

FreeTDS is an open source driver for the TDS protocol, this is the protocol used to talk to the MSSQL SQLServer database.

4.3.2 Installation Guide

Follow the remaining section in this document to prepare your RHEL operating system for to run the Peek Platform.

The instructions on this page don't install the peek platform, that's done later.

4.3.3 Install Red Hat Linux Server 7.7 OS

This section installs the Red Hat Linux Server 7.7 64bit operating system.

Create VM

Create a new virtual machine with the following specifications

- 4 CPUs
- 8gb of ram
- 60gb of disk space

Install OS

Download the RHEL ISO **Red Hat Enterprise Linux 7.7 Binary DVD** from:

[Download RHEL](#)

Mount the ISO in the virtual machine and start the virtual machine.

Note: Run through the installer manually, do not let your virtual machine software perform a wizard or express install.

Starting Off

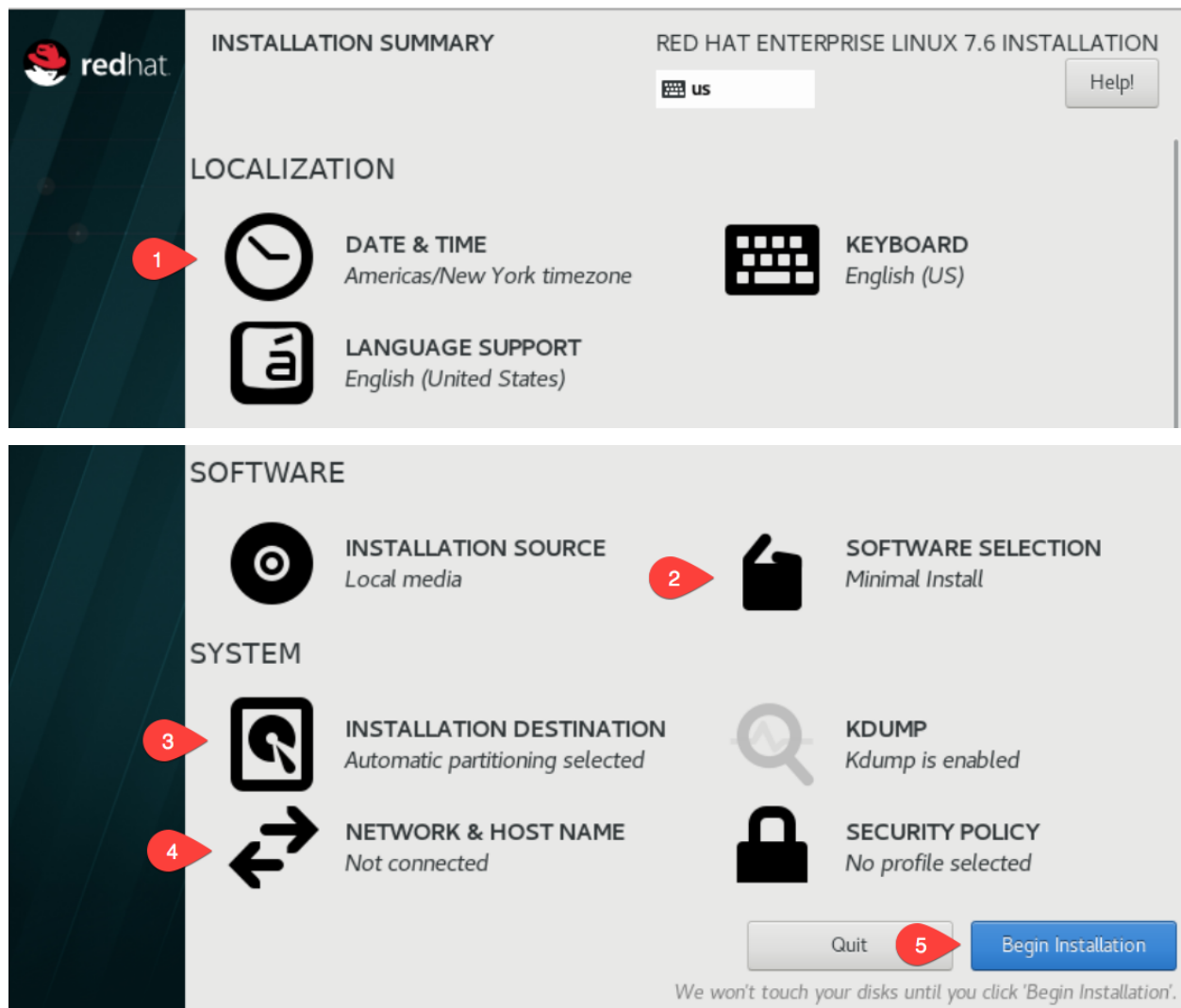
At the **Red Hat Enterprise Linux 7.7 installer boot menu** screen, select:

Install Red Hat Enterprise Linux 7.7

At the language selection screen, select:

English

Next you will see a screen that lets you jump to any area to configure. The areas that need attention are numbered and explained in the following sections.



#1 Goto the **DATE & TIME** screen, select the appropriate time location.

DATE & TIME

Done

RED HAT ENTERPRISE LINUX 7.4 INSTALLATION

us

Help!

Region: Australia

City: Melbourne

Network Time ON

19:11 PM

24-hour

AM/PM

21

06

18

⚠ You have no working NTP server configured

#2 Goto the **SOFTWARE SELECTION** screen, select **Minimal Install** or **Server with GUI** if you'd like a GUI.

SOFTWARE SELECTION

Done

RED HAT ENTERPRISE LINUX 7.4 INSTALLATION

us

Help! (F1)

Base Environment

☒ **Minimal Install**
Basic functionality.
 ☐ **Infrastructure Server**
Server for operating network infrastructure services.
 ☐ **File and Print Server**
File, print, and storage server for enterprises.
 ☐ **Basic Web Server**
Server for serving static and dynamic internet content.
 ☐ **Virtualization Host**
Minimal virtualization host.
 ☐ **Server with GUI**
Server for operating network infrastructure services, with a GUI.

Add-Ons for Selected Environment

☐ **Debugging Tools**
Tools for debugging misbehaving applications and diagnosing performance problems.
 ☐ **Compatibility Libraries**
Compatibility libraries for applications built on previous versions of Red Hat Enterprise Linux.
 ☐ **Development Tools**
A basic development environment.
 ☐ **Security Tools**
Security tools for integrity and trust verification.
 ☐ **Smart Card Support**
Support for using smart card authentication.

#3 Goto the **INSTALLATION DESTINATION** screen

The following partitioning is recommended for DEV peek virtual machines.

Select:

I will configure partitioning.

INSTALLATION DESTINATION Done **2** RED HAT ENTERPRISE LINUX 7.4 INSTALLATION us Help!

Device Selection

Select the device(s) you'd like to install to. They will be left untouched until you click on the main menu's "Begin Installation" button.

Local Standard Disks

50 GiB

VMware, VMware Virtual S

sda / 992.5 KiB free

Disks left unselected here will not be touched.

Specialized & Network Disks

Add a disk...

Disks left unselected here will not be touched.

Other Storage Options

Partitioning

☐ Automatically configure partitioning. ☒ **1** I will configure partitioning.

☐ I would like to make additional space available.

[Full disk summary and boot loader...](#) 1 disk selected; 50 GiB capacity; 992.5 KiB free [Refresh...](#)

Select Done.

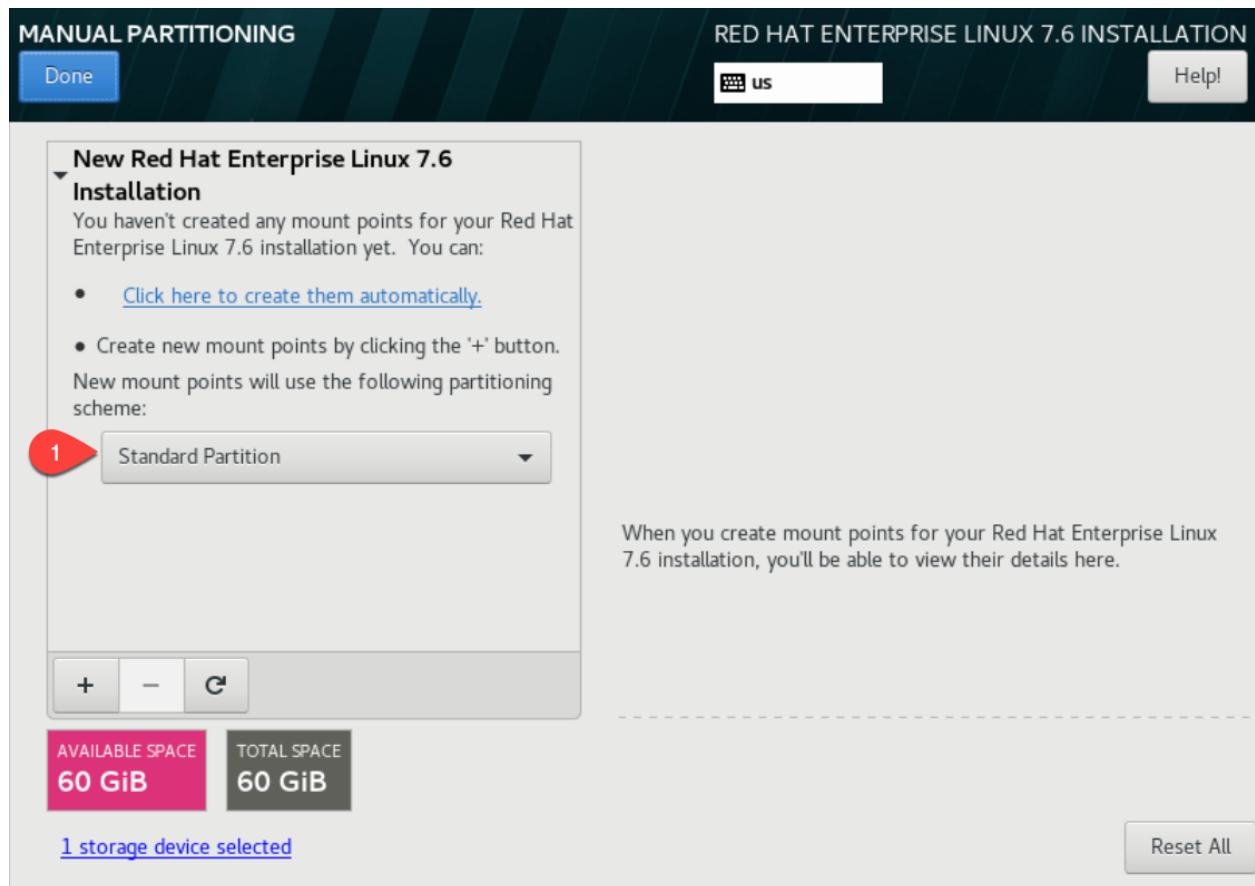
Partition Table

We'll be creating three partitions, */boot*, */* and *swap*. For a heavily used production server you may want to create more virtual disks and separate out */var*, */home*, and */tmp*. With one file system per disk.

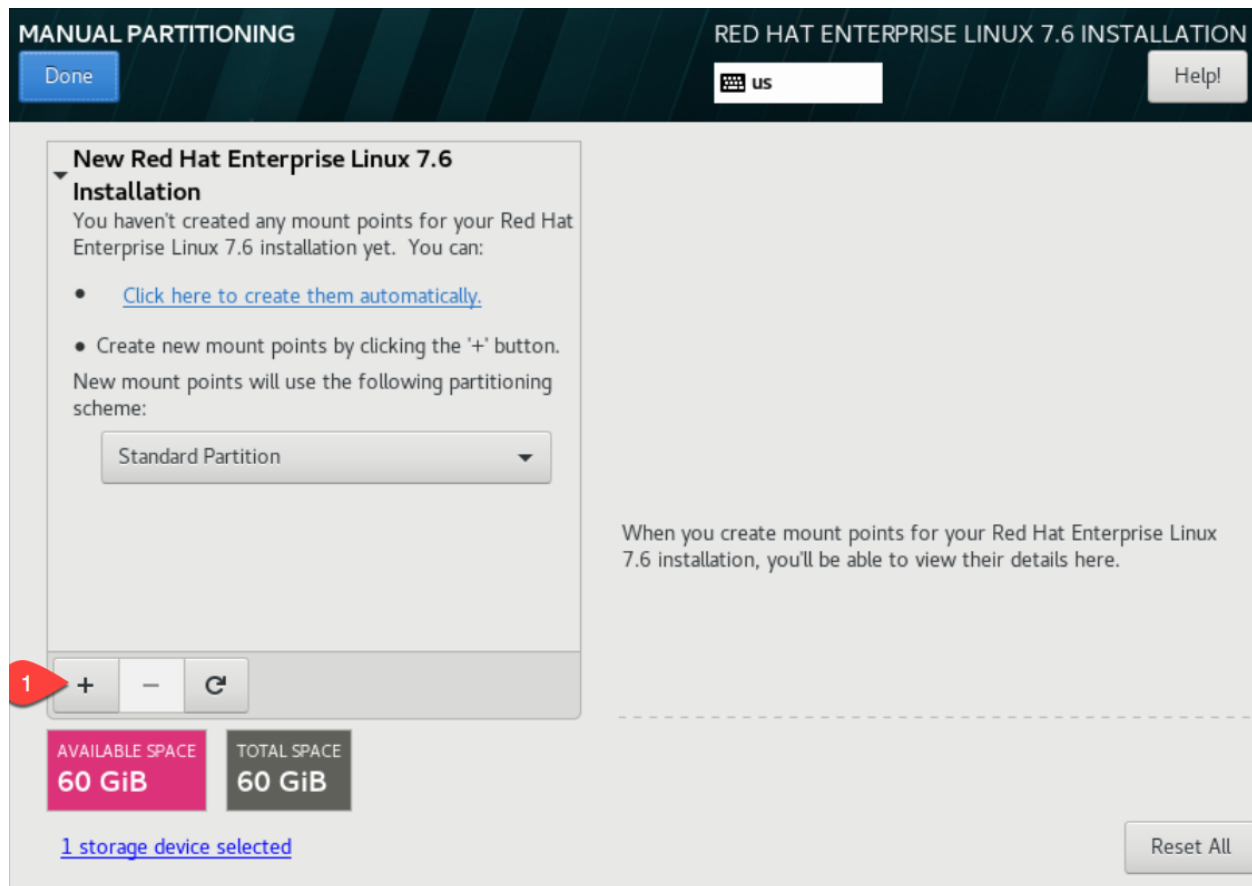
Having one file system per disk allows VM software to easily expand the disk and filesystem as required.

Select **Standard Partition**

Again, This is to allow the virtual machine software to expand the DEV server disks more easily.



Add the partitions, for each partition, click the plus.



Set the Mount Point to **/boot**

Set the size to **1g**

Click **Add mount point**

ADD A NEW MOUNT POINT

More customization options are available after creating the mount point below.

Mount Point:

Desired Capacity:

Set the Mount Point to **swap**

Set the size to **8g**

Click **Add mount point**

ADD A NEW MOUNT POINT

More customization options are available after creating the mount point below.

Mount Point: ▼

Desired Capacity:

Set the Mount Point to /

Set the size to **100%**

Click **Add mount point**

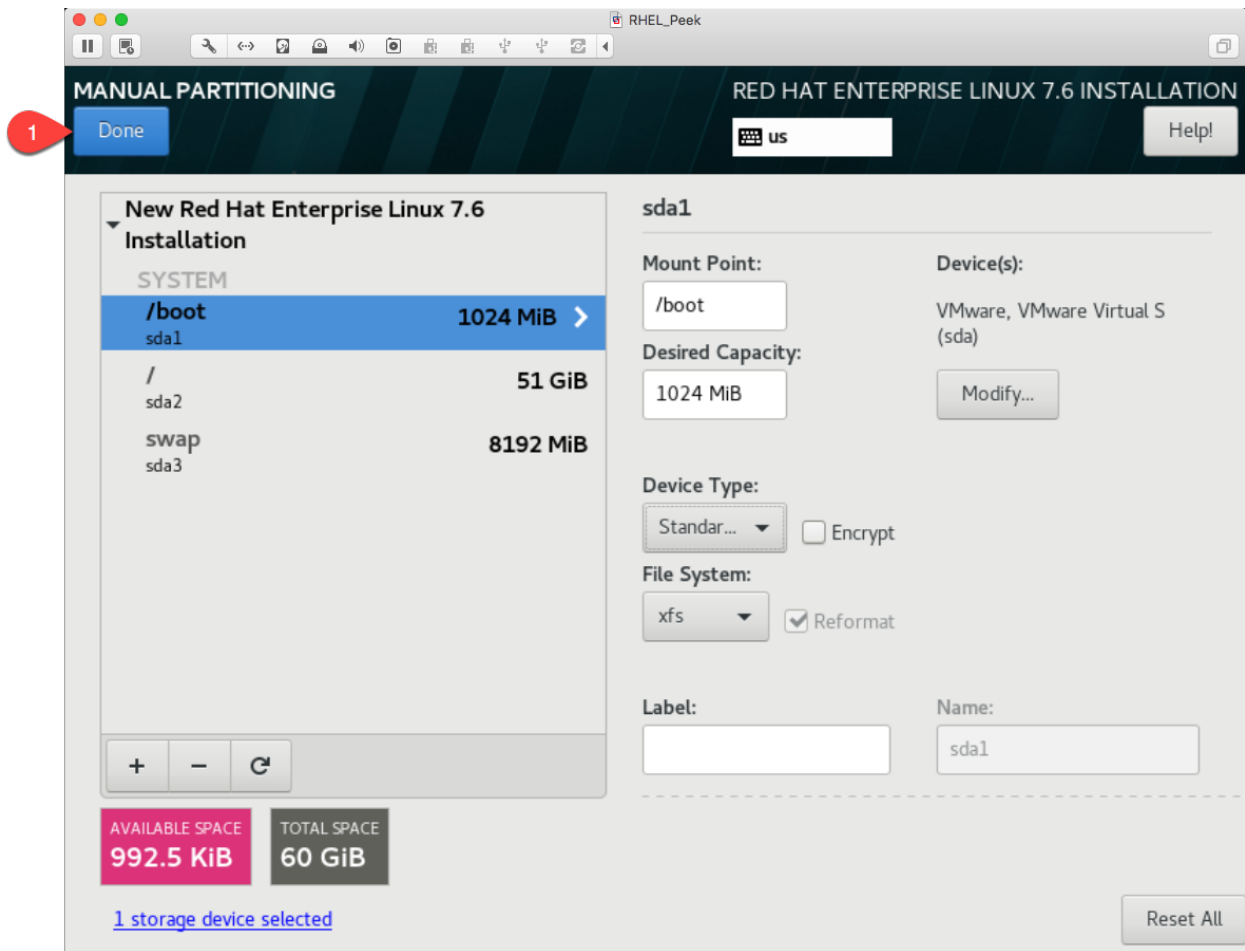
ADD A NEW MOUNT POINT

More customization options are available after creating the mount point below.

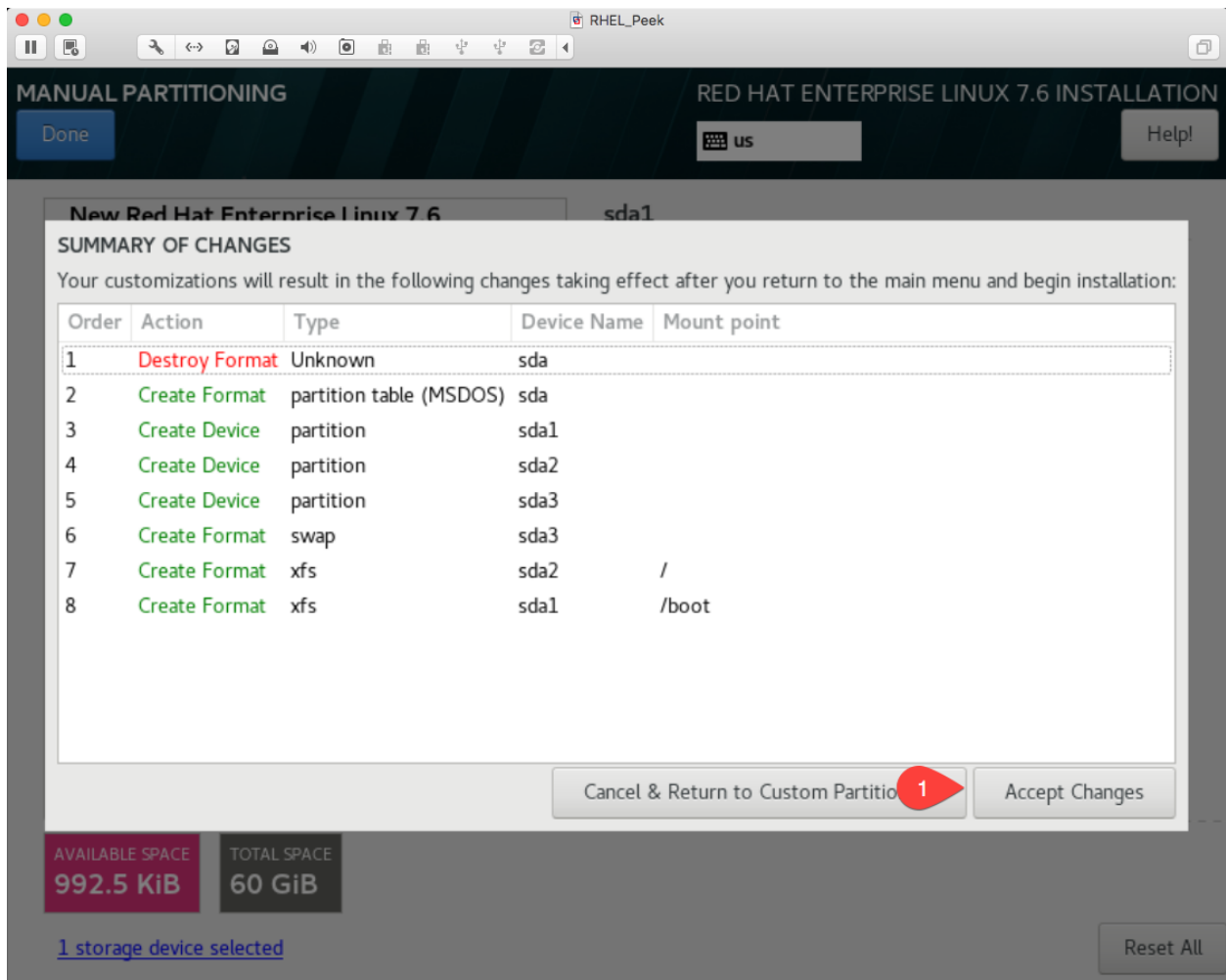
Mount Point:

Desired Capacity:

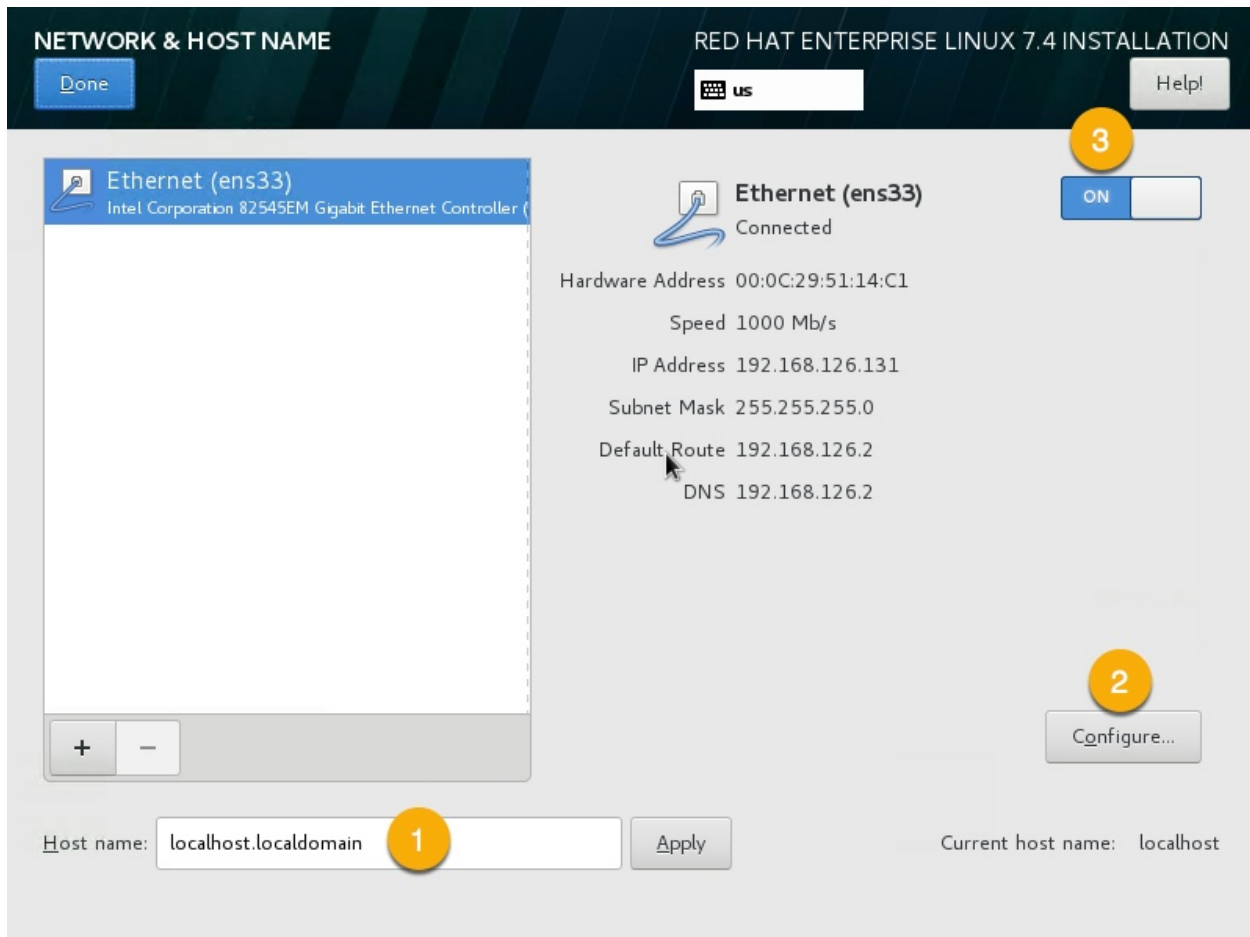
You should have a partition layout as follows, Click **Done**



Click **Accept Changes**



#4 Goto **NETWORK & HOST NAME** screen,



1. Enter your desired hostname, for example

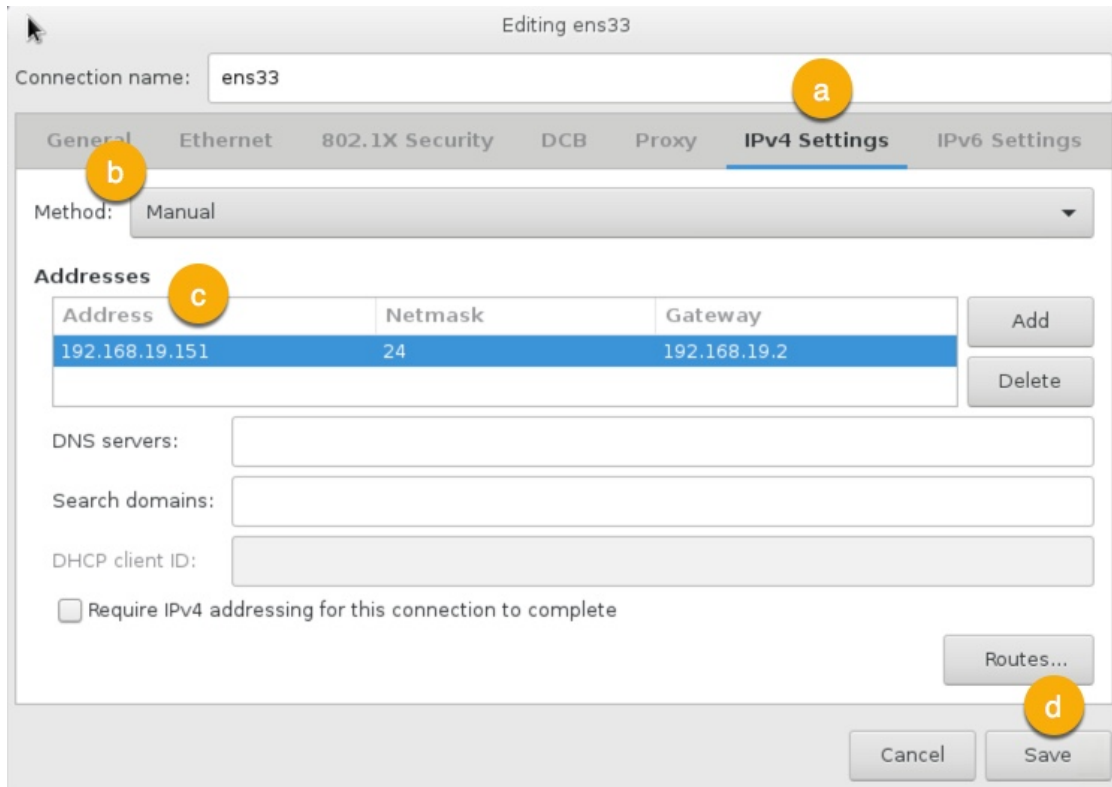
peek.localdomain

2. Turn on the Ethernet connection, this will get a DHCP IP Address.

Note: Make note of the DHCP IP Address

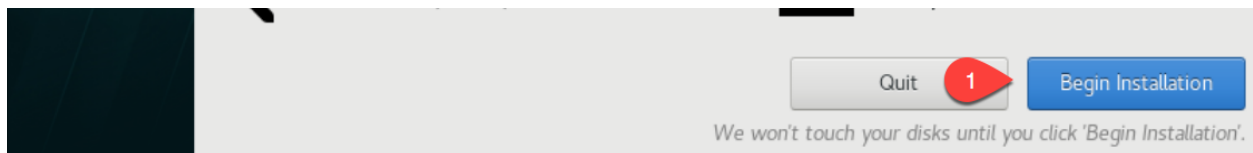
Otherwise, Configure a static IP address,

- a. Goto IPv4 Settings tab,
- b. Set Method to *Manual*,
- c. Add static IP address,
- d. Save.



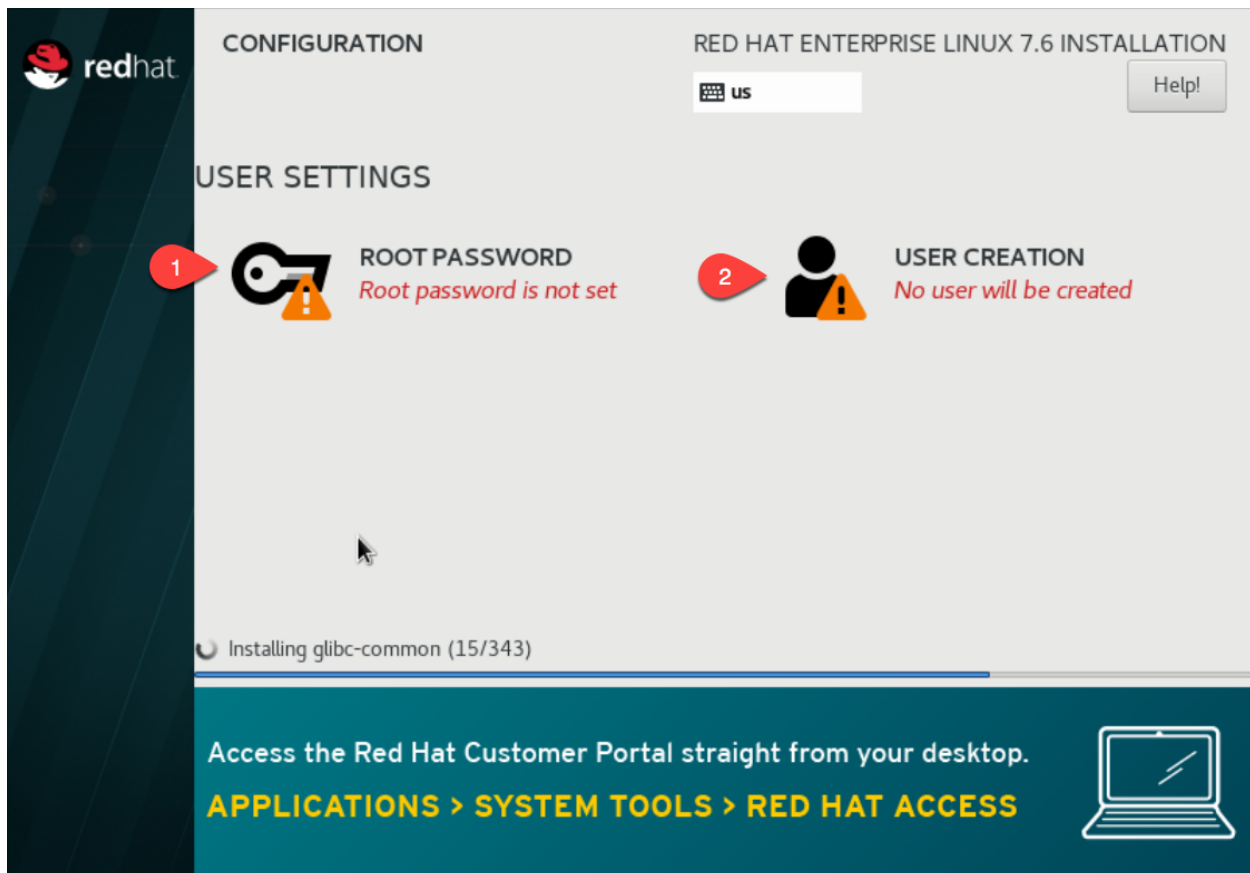
Select **DONE** review the **SUMMARY OF CHANGES**

Click **BEGIN INSTALLATION**



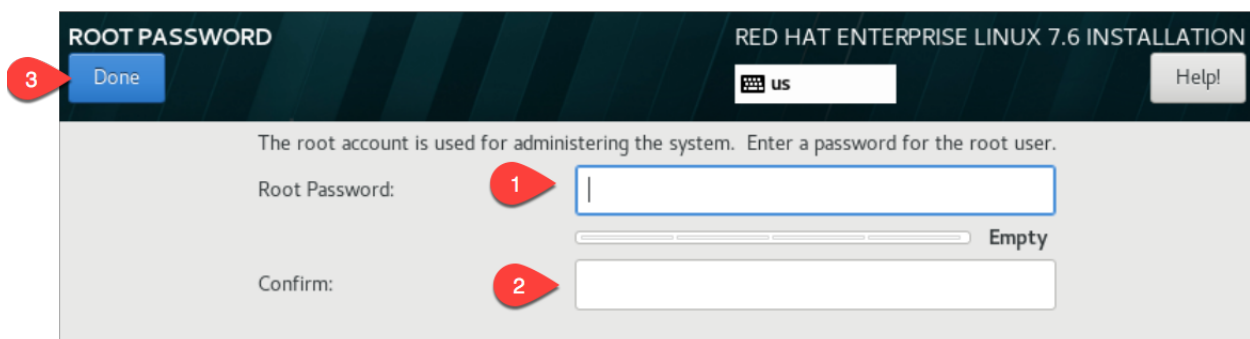
While RHEL is installing, further installation steps need to be completed.

Configure the **ROOT PASSWORD** and the **USER CREATION**



Configure the root password of the new RHEL VM.

Enter the root password twice, then click **Done**



Create the **peek** user as follows.

CREATE USER RED HAT ENTERPRISE LINUX 7.6 INSTALLATION

Done us Help!

1 Full name Peek Platform

2 User name peek

Tip: Keep your user name shorter than 32 characters and do not use spaces.

3 ☒ Make this user administrator

☒ Require a password to use this account

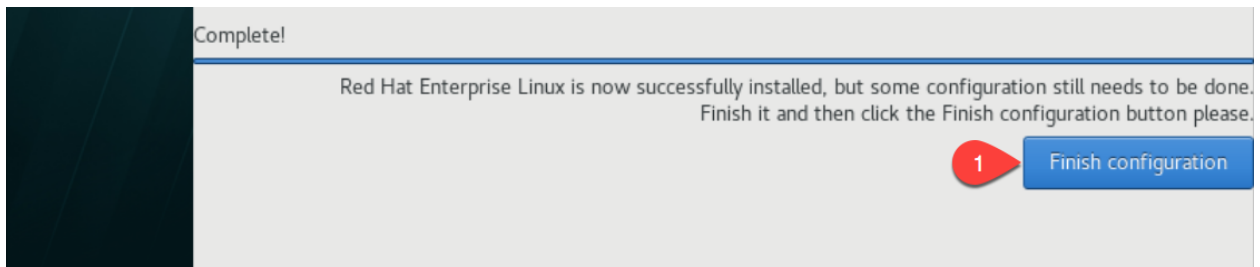
4 Password

5 Confirm password

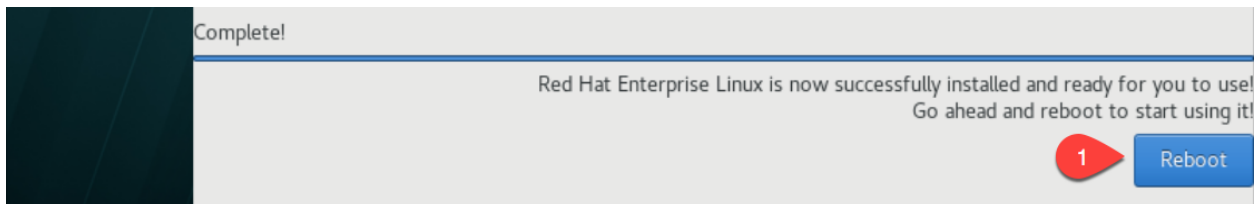
Weak

Advanced...

Click **Finish configuration**



Click **Reboot**



After the server has rebooted, disconnect and remove the RHEL ISO from DVD drive in the VM software.

The OS installation is now complete.

4.3.4 Login as Peek

Login to the RHEL VM as the `peek` user, either via SSH, or the graphical desktop if it's installed.

Important: All steps after this point assume you're logged in as the `peek` user.

4.3.5 Registering RHEL

The RHEL server must have access to the redhat repositories at `rhn.redhat.com` to install the required packages.

This section describes one way of registering a new RHEL server to a Redhat subscription. This is a paid subscription.

Run the following command to register the system. Replace `MY_RHN_USERNAME` with your redhat network username.

```
sudo date
# enter the password for peek

sudo subscription-manager register --username MY_RHN_USERNAME
# Enter the password for the RHN account
```

List the subscriptions, and select a pool.

```
sudo subscription-manager list --available | grep Pool
```

Subscribe to the pool. Replace `POOL_ID_FROM_ABOVE_COMMAND` with the Pool ID from the last command.

```
sudo subscription-manager subscribe --pool=POOL_ID_FROM_ABOVE_COMMAND
```

Test the subscription with a yum update, this will apply the latest updates.

```
sudo yum update -y
```

Note: If you want to remove the server from the pool, and unregister it, run the following.

```
sudo subscription-manager remove --all
sudo subscription-manager unregister
```

4.3.6 Removing IPv6 Localhost

Run the following command to ensure that localhost does not resolve to `::1` as this effects the PostgreSQL connection.

```
F=/etc/sysctl.conf
cat | sudo tee $F <<EOF
# Disable IPv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
EOF
sudo sysctl -p
sudo sed -i '/:::1/d' /etc/hosts
```

4.3.7 Installing General Prerequisites

This section installs the OS packages required.

Note: Run the commands in this step as the *peek* user.

To begin, make sure that all the packages currently installed on your RHEL system are updated to their latest versions:

```
sudo yum update -y
```

Install the C Compiler package, used for compiling python or VMWare tools, etc:

```
PKG="gcc gcc-c++ kernel-devel make"
sudo yum install -y $PKG
```

Install rsync:

```
PKG="rsync"
PKG="$PKG unzip"
PKG="$PKG wget"
PKG="$PKG git"
PKG="$PKG bzip2"

sudo yum install -y $PKG
```

Install the Python build dependencies:

```
PKG="curl git m4 ruby texinfo bzip2-devel libcurl-devel"
PKG="$PKG expat-devel ncurses-libs zlib-devel gmp-devel"
PKG="$PKG openssl openssl-devel"
sudo yum install -y $PKG
```

Install the PostgreSQL build dependencies:

```
PKG="bison flex"
PKG="$PKG readline-devel openssl-devel python-devel"
sudo yum install -y $PKG
```

Install Libs that some python packages link to when they install.

```
# For the cryptography package
PKG="libffi-devel"

sudo yum install -y $PKG
```

Install Libs required for LDAP.

```
PKG="openldap-devel"

sudo yum install -y $PKG
```

Install Libs that database access python packages link to when they install:

Warning: These packages are not from the Redhat Network.

```
FEDORA_PACKAGES="https://dl.fedoraproject.org/pub/epel/7/x86_64/Packages"

# For Shapely and GEOAlchemy
PKG="${FEDORA_PACKAGES}/g/geos-3.4.2-2.el7.x86_64.rpm"
PKG="$PKG ${FEDORA_PACKAGES}/g/geos-devel-3.4.2-2.el7.x86_64.rpm"

# For the SQLite python connector
PKG="$PKG ${FEDORA_PACKAGES}/l/libsqlite3x-20071018-20.el7.x86_64.rpm"
PKG="$PKG ${FEDORA_PACKAGES}/l/libsqlite3x-devel-20071018-20.el7.x86_64.rpm"

sudo yum install -y $PKG
```

Install Libs that the oracle client requires:

```
# For LXML and the Oracle client
PKG="libxml2 libxml2-devel"
PKG="$PKG libxslt libxslt-devel"
PKG="$PKG libaio libaio-devel"

sudo yum install -y $PKG
```

Cleanup the downloaded packages:

```
sudo yum clean all
```

4.3.8 Installing VMWare Tools (Optional)

This section installs VMWare tools. The compiler tools have been installed from the section above.

In the VMWare software, find the option to install VMWare tools.

Mount and unzip the tools:

```
sudo rm -rf /tmp/vmware-*
sudo mount /dev/sr0 /mnt
sudo tar -xzf /mnt/VM*.gz -C /tmp
sudo umount /mnt
```

Install the tools with the default options:

```
cd /tmp/vmware-tools-distrib
sudo ./vmware-install.pl -f -d
```

Cleanup the tools install:

```
sudo rm -rf /tmp/vmware-*
```

Reboot the virtual machine:

```
sudo shutdown -r now
```

Note: Keep in mind, that if the static IP is not set, the IP address of the VM may change, causing issues when reconnecting with SSH.

4.3.9 Update Firewall

Allow Peek through the firewall and port forward to the non-privileged port

```
# Peek Mobile website
sudo firewall-cmd --add-forward-port=port=8000:proto=tcp:toport=8000

# Peek Desktop website
sudo firewall-cmd --add-forward-port=port=8002:proto=tcp:toport=8002

# Peek Admin web site
sudo firewall-cmd --add-forward-port=port=8010:proto=tcp:toport=8010

# Persist the rules
sudo firewall-cmd --runtime-to-permanent
```

4.3.10 Preparing .bashrc

Open ~/.bashrc insert the following at the start:

```
##### SET THE PEEK ENVIRONMENT #####
# Setup the variables for PYTHON and POSTGRESQL
export PEEK_PY_VER="3.9.1"
export PEEK_TSDB_VER="1.7.4"
export PGDATA=~peek/pgdata/12

export PATH="$HOME/opt/bin:$PATH"
export LD_LIBRARY_PATH="$HOME/opt/lib:$LD_LIBRARY_PATH"

# Set the variables for the platform release
# These are updated by the deploy script
export PEEK_ENV=""
[ -n "${PEEK_ENV}" ] && export PATH="${PEEK_ENV}/bin:$PATH"
```

Warning: Restart your terminal to get the new environment.

4.3.11 Compile and Install Python 3.9.1

The Peek Platform runs on Python. These instructions download, compile and install the latest version of Python.

Download and unarchive the supported version of Python:

```
cd
source .bashrc
wget https://github.com/python/cpython/archive/v${PEEK_PY_VER}.zip
unzip v${PEEK_PY_VER}.zip
cd cpython-${PEEK_PY_VER}
```

Configure the build:

```
./configure --prefix=/home/peek/opt/ --enable-optimizations --enable-shared
```

Make and Make install the software:

```
make install
```

Cleanup the download and build dir:

```
cd
rm -rf cpython-${PEEK_PY_VER}
rm v${PEEK_PY_VER}.zip
```

Symlink the python3 commands so they are the only ones picked up by path:

```
cd /home/peek/opt/bin
ln -s pip3 pip
ln -s python3 python
cd
```

Test that the setup is working:

```
RED='\033[0;31m'
GREEN='\033[0;32m'
NC='\033[0m' # No Color

SHOULD_BE="/home/peek/opt/bin/python"
if [ `which python` == ${SHOULD_BE} ]
then
    echo -e "${GREEN}SUCCESS${NC} The python path is right"
else
    echo -e "${RED}FAIL${NC} The python path is wrong, It should be ${SHOULD_BE}"
fi

SHOULD_BE="/home/peek/opt/bin/pip"
if [ `which pip` == ${SHOULD_BE} ]
then
    echo -e "${GREEN}SUCCESS${NC} The pip path is right"
else
    echo -e "${RED}FAIL${NC} The pip path is wrong, It should be ${SHOULD_BE}"
fi
```

Upgrade pip:

```
pip install --upgrade pip
```

synerty-peek is deployed into python virtual environments. Install the virtualenv python package:

```
pip install virtualenv
```

The Wheel package is required for building platform and plugin releases:

```
pip install wheel
```

4.3.12 Install PostgreSQL

Install the relational database Peek stores its data in. This database is PostgreSQL 12.

Note: Run the commands in this step as the peek user.

Download the PostgreSQL source code

```
PEEK_PG_VER=12.5
SRC_DIR="$HOME/postgresql-${PEEK_PG_VER}"

# Remove the src dir and install file
rm -rf ${SRC_DIR} || true
cd $HOME

wget https://ftp.postgresql.org/pub/source/v${PEEK_PG_VER}/postgresql-${PEEK_PG_VER}.
↪tar.bz2
tar xjf postgresql-${PEEK_PG_VER}.tar.bz2

cd ${SRC_DIR}
```

Configure and build PostGresQL

```
export CPPFLAGS="-I`echo $HOME/opt/include/python*m` "
export LDFLAGS="-L$HOME/opt/lib "

./configure \
    --disable-debug \
    --prefix=$HOME/opt \
    --enable-thread-safety \
    --with-openssl \
    --with-python

make -j4

make install-world

# this is required for timescale to compile
cp ${SRC_DIR}/src/test/isolation/pg_isolation_regress ~/opt/bin
```

Remove install files to clean up the home directory

```
# Remove the src dir and install file
cd
rm -rf ${SRC_DIR}*
```

Initialise a PostgreSQL database

```
#Refresh .bashrc so initdb can find postgres
source .bashrc

initdb --pgdata=$HOME/pgdata/12 --auth-local=trust --auth-host=md5
```

Tune the postgresql.conf to increase the maximum number of connections allowed

```
F="$HOME/pgdata/12/postgresql.conf"

sed -i 's/max_connections = 100/max_connections = 200/g' $F
```


Make PostgreSQL a service :

Note: This will require sudo permissions

Run the following command

```
touch postgresql-12.service

F=postgresql-12.service

cat <<"EOF" | sed "s,\$HOME,`echo ~peek`,g" > $F
[Unit]
Description=PostgreSQL 12 database server
After=syslog.target
After=network.target

[Service]
Type=forking
User=peek
Group=peek

# Location of database directory
Environment=PGDATA=$HOME/pgdata/12

# Disable OOM kill on the postmaster
OOMScoreAdjust=-1000
Environment=PG_OOM_ADJUST_FILE=/proc/self/oom_score_adj
Environment=PG_OOM_ADJUST_VALUE=0

ExecStart=$HOME/opt/bin/pg_ctl -D ${PGDATA} start
ExecStop=$HOME/opt/bin/pg_ctl -D ${PGDATA} stop
ExecReload=/bin/kill -HUP $MAINPID
KillMode=mixed
KillSignal=SIGINT

# Do not set any timeout value, so that systemd will not kill postmaster
# during crash recovery.
TimeoutSec=0

[Install]
WantedBy=multi-user.target
EOF

sudo mv $F /usr/lib/systemd/system/postgresql-12.service
```

Reload the daemon

```
systemctl daemon-reload
```

Install CMake

Download CMake source code

```
PEEK_CMAKE_VER=3.19.2
SRC_DIR="$HOME/CMake-${PEEK_CMAKE_VER}"
wget https://github.com/Kitware/CMake/archive/v${PEEK_CMAKE_VER}.zip

unzip v${PEEK_CMAKE_VER}.zip
cd ${SRC_DIR}
```

Compile CMake from source

```
./configure --prefix=$HOME/opt

make -j6 install

# Remove the src dir and install file
cd
rm -rf ${SRC_DIR}*
rm v${PEEK_CMAKE_VER}.zip
```

Install PostgreSQL Timescaledb

Next install timescaledb, this provides support for storing large amounts of historical data.

www.timescale.com

Download the timescaledb source code

```
PEEK_TSDB_VER=1.7.4

cd
wget https://github.com/timescale/timescaledb/archive/${PEEK_TSDB_VER}.zip
unzip ${PEEK_TSDB_VER}.zip
cd timescaledb-${PEEK_TSDB_VER}
```

Install the packages:

```
export CPPFLAGS=`pg_config --cppflags`
export LDFLAGS=`pg_config --ldflags`

# Bootstrap the build system
./bootstrap -DAPACHE_ONLY=1

# To build the extension
cd build && make

# To install
make install

# Cleanup the source code
cd
rm -rf ${PEEK_TSDB_VER}.zip
rm -rf timescaledb-${PEEK_TSDB_VER}
```

Add the timescale repository:

```
curl -s https://packagecloud.io/install/repositories/timescale/timescaledb/script.rpm.  
↪sh | sudo bash
```

Install timescaledb-tune:

```
sudo yum install -y timescaledb-tools-0.10.0-0.e17.x86_64
```

Tune the database:

```
PGVER=12  
FILE="$HOME/pgdata/${PGVER}/postgresql.conf"  
timescaledb-tune -quiet -yes -conf-path ${FILE} -pg-version ${PGVER}
```

Start PostgreSQL:

```
systemctl enable postgresql-12 --now
```

Finish PostgreSQL Setup

Finish configuring and starting PostgreSQL.

Allow the peek OS user to login to the database as user peek with no password

```
F=$HOME/pgdata/12/pg_hba.conf  
cat | sudo tee $F <<EOF  
# TYPE DATABASE USER ADDRESS METHOD  
local all peek trust  
  
# "local" is for Unix domain socket connections only  
local all all peer  
# IPv4 local connections:  
host all all 127.0.0.1/32 md5  
# IPv6 local connections:  
host all all ::1/128 md5  
EOF
```

Create the database:

```
createdb -O peek peek
```

Set the PostgreSQL peek users password:

```
psql -d peek -U peek <<EOF  
\password  
\q  
EOF  
  
# Set the password as "PASSWORD" for development machines  
# Set it to a secure password from https://xkpasswd.net/s/ for production
```

Note:

If you already have a database, you may now need to upgrade the `timescale` extension.

```
psql peek <<EOF ALTER EXTENSION timescaledb UPDATE; EOF
```

Cleanup traces of the password:

```
[ ! -e ~/.psql_history ] || rm ~/.psql_history
```

Grant PostgreSQL Peek Permissions

The PostgreSQL server now runs parts of peaks python code inside the postgres/postmaster processes. To do this the postgres user needs access to peaks home directory where the peek software is installed.

Grant permissions

```
sudo chmod g+rx ~peek
```

4.3.13 Install Worker Dependencies

Install the parallel processing queue we use for the peek-worker-service tasks.

Note: Run the commands in this section as the *peek* user.

Install redis:

```
ATOMICORP_SITE="https://www6.atomicorp.com/channels/atomic/centos/7/x86_64/RPMS"

# redis dependencies
PKG="${ATOMICORP_SITE}/jemalloc-3.6.0-1.el7.art.x86_64.rpm"

# redis
PKG="$PKG ${ATOMICORP_SITE}/redis-3.0.7-4.el7.art.x86_64.rpm"

# install redis and dependencies
sudo yum install -y $PKG
```

Enable the Redis service:

```
sudo systemctl enable redis.service
sudo systemctl restart redis.service
```

Install rabbitmq:

```
# install erlang
curl -s https://packagecloud.io/install/repositories/rabbitmq/erlang/script.rpm.sh |  sudo bash

# install erlang
sudo yum install -y erlang

# Set rabbitmq repository
curl -s https://packagecloud.io/install/repositories/rabbitmq/rabbitmq-server/script.  
rpm.sh | sudo bash

# install rabbitmq
sudo yum install -y rabbitmq-server
```

Enable the RabbitMQ service:

```
sudo systemctl enable rabbitmq-server.service
sudo systemctl restart rabbitmq-server.service
```

Cleanup the downloaded packages:

```
sudo yum clean all
```

Enable the RabbitMQ management plugins:

```
F="/var/lib/rabbitmq/.erlang.cookie"; [ ! -f $F ] || rm -f $F
sudo rabbitmq-plugins enable rabbitmq_mqtt
sudo rabbitmq-plugins enable rabbitmq_management
sudo systemctl restart rabbitmq-server.service
```

Increase the size of the redis client queue

```
BEFORE="client-output-buffer-limit pubsub 64mb 16mb 90"
AFTER="client-output-buffer-limit pubsub 32mb 8mb 60"
sudo sed -i "s/${BEFORE}/${AFTER}/g" /etc/redis.conf

sudo systemctl restart redis
```

4.3.14 Install Oracle Client (Optional)

The oracle libraries are optional. Install them where the agent runs if you are going to interface with an oracle database.

Open `~/.bashrc` insert the following at the start:

```
# Setup the variables for ORACLE
export LD_LIBRARY_PATH="/home/peek/oracle/instantclient_21_1:$LD_LIBRARY_PATH"
export ORACLE_HOME="/home/peek/oracle/instantclient_21_1"
```

Source the new profile to get the new variables:

```
source ~/.bashrc
```

Make the directory where the oracle client will live

```
mkdir /home/peek/oracle
```

Download the following from oracle.

The version used in these instructions is **21.1.0.0.0**.

1. Download the ZIP “Basic Package” `instantclient-basic-linux.x64-21.1.0.0.0.zip` from <http://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html>
2. Download the ZIP “SDK Package” `instantclient-sdk-linux.x64-21.1.0.0.0.zip` from <http://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html>

Copy these files to `/home/peek/oracle` on the peek server.

Extract the files.

```
cd ~/oracle
unzip instantclient-basic-linux.x64-21.1.0.0.0.zip*
unzip instantclient-sdk-linux.x64-21.1.0.0.0.zip*
```

4.3.15 Install FreeTDS (Optional)

FreeTDS is an open source driver for the TDS protocol, this is the protocol used to talk to a MSSQL SQLServer database.

Peek needs this installed if it uses the `pymssql` python database driver, which depends on FreeTDS.

Open `~/.bashrc` insert the following at the start:

```
# Setup the variables for FREE TDS
export LD_LIBRARY_PATH="/home/peek/freetds:$LD_LIBRARY_PATH"
```

Warning: Restart your terminal you get the new environment.

Install FreeTDS:

```
PKG="https://dl.fedoraproject.org/pub/epel/7/x86_64/Packages/f/freetds-libs-1.1.20-1.
↪el7.x86_64.rpm"
PKG="$PKG https://dl.fedoraproject.org/pub/epel/7/x86_64/Packages/f/freetds-1.1.20-1.
↪el7.x86_64.rpm"
PKG="$PKG https://dl.fedoraproject.org/pub/epel/7/x86_64/Packages/f/freetds-devel-1.1.
↪20-1.el7.x86_64.rpm"
sudo yum install -y $PKG
```

Create file `freetds.conf` in `~/freetds` and populate with the following:

```
mkdir ~/freetds
cat > ~/freetds/freetds.conf <<EOF

[global]
    port = 1433
    instance = peek
    tds version = 7.4

EOF
```

If you want to get more debug information, add the dump file line to the `[global]` section Keep in mind that the dump file takes a lot of space.

```
[global]
    port = 1433
    instance = peek
    tds version = 7.4
    dump file = /tmp/freetds.log
```

4.3.16 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

4.4 Setup OS Requirements Debian

This section describes how to perform the setup for Debian Linux 10. The Peek platform is designed to run on Linux. Please read through all of the documentation before commencing the installation procedure.

Note: These instructions are for Debian 10, AKA Buster

4.4.1 Installation Objective

This Installation Guide contains specific Debian Linux 10 operating system requirements for the configuring of synerty-peek.

Required Software

Some of the software to be installed requires internet access. For offline installation some steps are required to be installed on another online server for the files to be packaged and transferred to the offline server.

Below is a list of all the required software:

- Python 3.9.x
- Postgres 12.x

Suggested Software

The following utilities are often useful.

- rsync
- git
- unzip

Optional Software

- Oracle Client

Installing Oracle Libraries is required if you intend on installing the peek agent. Instruction for installing the Oracle Libraries are in the Online Installation Guide.

- FreeTDS

FreeTDS is an open source driver for the TDS protocol, this is the protocol used to talk to the MSSQL SQLServer database.

4.4.2 Installation Guide

Follow the remaining section in this document to prepare your debian operating system for to run the Peek Platform.

The instructions on this page don't install the peek platform, that's done later.

4.4.3 Install Debian 10 OS

This section installs the Debian 10 64bit Linux operating system.

Create VM

Create a new virtual machine with the following specifications

- 4 CPUs
- 8gb of ram
- 60gb of disk space

Install OS

Download the debian ISO, navigate to the following site and click **amd64** under **netinst CD image**

[Download Debian](#)

Mount the ISO in the virtual machine and start the virtual machine.

Run through the installer manually, do not let your virtual machine software perform a wizard or express install.

Starting Off

At the **Debian GNU/Linux installer boot menu** screen, select:

Install

At the **Select a language** screen, select:

English

At the **Select your location** screen, select the appropriate location.

At the **Configure the keyboard** screen, select the appropriate keyboard, or leave as default.

At the **Configure the network** screen, enter your desired hostname or:

peek

At the **Configure the network** screen, enter your desired domain, or:

localdomain

At the **Setup users and passwords screen**, watch for the following prompts, replace `<root_password>` and `<peek_password>` with your desired passwords.

Table 1: Setup users and passwords screen prompts

Prompt	Enter :
Root password	<root_password>
Re-enter password to verify	<root_password>
Full name for the new user	Peek Platform
Username for your account	peek
Choose a password for the new user:	<peek_password>
Re-enter password to verify:	<peek_password>

On the **Configure the clock** screen, select your desired timezone.

Partition Table

On the **Partition disks** screen, select:

```
Manual
```

Then, select the disk, it will look similar to:

```
SCSI3 (0,0,0) (sda) - 32.2 GB VMware ...
```

Then it will prompt to **Create new empty partition table on this device?**, select:

```
<Yes>
```

We'll be creating three partitions, /boot / and swap. For a heavily used production server you may want to create more virtual disks and separate out /var, /home, and /tmp. With one file system per disk.

Having one file system per disk removes the need for the overhead of LVM, and the VM software can still expand the disk and filesystem as required.

/boot

Select the following disk from the menu:

```
pri/log *** GB  FREE SPACE
```

Enter the following responses to the prompts

Table 2: /boot partition prompts part1

Prompt	Enter :
How to user this free space	Create a new partition
New partition size	500m
Type for the new partition	Primary
Location for the new Partition	Beginning

At the **Partition Settings** prompt, enter the following:

Table 3: /boot partition prompts part2

Prompt	Enter :
Use as:	Ext2 file system
Mount point	/boot
Done setting up the partition	

swap

Select the following disk from the menu:

```
pri/log *** GB  FREE SPACE
```

Enter the following responses to the prompts

Table 4: swap partition prompts part1

Prompt	Enter :
How to user this free space	Create a new partition
New partition size	4g
Type for the new partition	Primary
Location for the new Partition	Beginning

At the **Partition Settings** prompt, enter the following:

Table 5: swap partition prompts part2

Prompt	Enter :
Use as:	swap
Done setting up the partition	

/ (root)

The root file system is created at the end of the disk, ensuring that if we use the VM software to expand the virtual disk, this is the file system that will be expanded.

The default guided install doesn't do this.

Select the following disk from the menu:

```
pri/log *.* GB  FREE SPACE
```

Enter the following responses to the prompts

Table 6: swap partition prompts part1

Prompt	Enter :
How to user this free space	Create a new partition
New partition size	100%
Type for the new partition	Primary
Location for the new Partition	Beginning

At the **Partition Settings** prompt, enter the following:

Table 7: swap partition prompts part2

Prompt	Enter :
Use as	Ext4 journaling file system
Mount point	/
Reserved blocks	1%
Done setting up the partition	

All done, select:

```
Finish partitioning and write changes to disk
```

At the **Write the changes to disk?** prompt, Select:

`<Yes>`

Finishing Up

On the **Configure the package manager** screen, select the location closest to you.

At the **Debian archive mirror**, select your preferred site.

At the **HTTP proxy information** prompt, select:

`<Continue>`

The installer will now download the package lists.

At the **Configure popularity-contest** screen, select:

`<No>`

Note: It'd be good to select `<Yes>`, but as Peek is an enterprise platform, it's most likely installed behind a corporate firewall.

At the **Software selection** screen, select the following, and deselect all the other options:

- SSH server
- standard system utilities

Optionally, select a desktop environment, Peek doesn't require this. "MATE" is recommended if one is selected.

The OS will now install, it will take a while to download and install the packages.

At the **Install the GRUB boot loader on a hard disk** screen, select:

`<Yes>`

At the **Device for boot loader installation** prompt, select:

`/dev/sda`

At the **Finish the installation** screen, select:

```
<Continue>
```

Deconfigure the Debian ISO from DVD drive in the VM software.

The OS installtion is now complete.

4.4.4 SSH Setup

SSH is this documentations method of working with the Peek Debian VM.

SSH clients are available out of the box with OSX and Linux. There are many options for windows users, This documentation recommends [MobaXterm](#) is used for windows as it also supports graphical file copying.

This document assumes users are familair with what is required to use the SSH clients for connecting to and copying files to the Peek VM.

If this all sounds too much, reinstall the Peek OS with a graphical desktop environment and use that instead of SSH.

Note: You will not be able to login as root via SSH by default.

Login to the console of the Peek Debian VM as root and install ifconfig with the following command:

```
apt-get install net-tools
```

Run the following command:

```
ifconfig
```

Make note of the ipaddress, you will need this to SSH to the VM. The IP addresss will be under **eth0**, second line, **inet addr**.

Install sudo with the following command:

```
apt-get install sudo
```

Give Peek sudo privielges with the following command:

```
usermod -a -G sudo peek
```

You must now logout from the root console.

4.4.5 Login as Peek

Login to the Debian VM as the `peek` user, either via SSH, or the graphical desktop if it's installed.

Important: All steps after this point assume you're logged in as the `peek` user.

4.4.6 Configure Static IP (Optional)

If this is a production server, it's more than likely that you want to assign a static IP to the VM, Here is how you do this.

Edit file `/etc/network/interfaces`

Find the section:

```
allow-hotplug eth0
iface eth0 inet dhcp
```

Replace it with:

```
auto eth0
iface eth0 inet static
    address <IPADDRESS>
    netmask <NETMASK>
    gateway <GATEWAY>
```

Edit the file `/etc/resolv.conf`, and update it.

1. Replace "localdomain" with your domain
2. Replace the IP for the `nameserver` with the IP of you DNS. For multiple name servers, use multiple `nameserver` lines.

```
domain localdomain
search localdomain
nameserver 172.16.40.2
```

4.4.7 Installing General Prerequisites

This section installs the OS packages required.

Note: Run the commands in this step as the `peek` user.

Install the C Compiler package, used for compiling python or VMWare tools, etc:

```
PKG="gcc make linux-headers-amd64"
sudo apt-get install -y $PKG
```

Install some utility packages:

```
PKG="rsync unzip wget git"

sudo apt-get install -y $PKG
```

Install the Python build dependencies:

```
PKG="build-essential curl git m4 ruby texinfo libbz2-dev libcurl4-openssl-dev"
PKG="$PKG libexpat-dev libncurses-dev zlib1g-dev libgmp-dev libssl-dev"
sudo apt-get install -y $PKG
```

Install the Postgres build dependencies:

```
PKG="bison flex"
PKG="$PKG libreadline-dev python-dev"
sudo apt-get install -y $PKG
```

Install Libs that some python packages link to when they install:

```
# For the cryptography package
PKG="libffi-dev"

sudo apt-get install -y $PKG
```

Install Libs required for LDAP:

```
PKG="libsasl2-dev libldap-common libldap2-dev"

sudo apt-get install -y $PKG
```

Install Libs that database access python packages link to when they install:

```
# For Shapely and GEOAlchemy
PKG="libgeos-dev libgeos-c1v5"

# For the PostGresQL connector
PKG="$PKG libpq-dev"

# For the SQLite python connector
PKG="$PKG libsqlite3-dev"

sudo apt-get install -y $PKG
```

Install Libs that the oracle client requires:

```
# For LXML and the Oracle client
PKG="libxml2 libxml2-dev"
PKG="$PKG libxslt1.1 libxslt1-dev"
PKG="$PKG libaio1 libaio-dev"

sudo apt-get install -y $PKG
```

Cleanup the downloaded packages

```
sudo apt-get clean
```

4.4.8 Installing VMWare Tools (Optional)

This section installs VMWare tools. The compiler tools have been installed from the section above.

In the VMWare software, find the option to install VMWare tools.

Mount and unzip the tools

```
sudo rm -rf /tmp/vmware-*
sudo mount /dev/sr0 /mnt
sudo tar xzf /mnt/VM*.gz -C /tmp
sudo umount /mnt
```

Install the tools with the default options

```
cd /tmp/vmware-tools-distrib
sudo ./vmware-install.pl -f -d
```

Cleanup the tools install

```
sudo rm -rf /tmp/vmware-*
```

Reboot the virtual machine.

```
sudo shutdown -r now
```

Keep in mind, that if the static IP is not set, the IP address of the VM may change, causing issues when reconnecting with SSH.

4.4.9 Preparing .bashrc

Warning: Open `~/ .bashrc` insert the following at the start, before:

```
# If not running interactively, don't do anything
```

If you do not place the below code before that line, it will not be parsed.

```
##### SET THE PEEK ENVIRONMENT #####
# Setup the variables for PYTHON and POSTGRESQL
export PEEK_PY_VER="3.9.1"
export PEEK_TSDB_VER="1.7.4"
export PGDATA=~peek/pgdata/12

export PATH="$HOME/opt/bin:$PATH"
export LD_LIBRARY_PATH="$HOME/opt/lib:$LD_LIBRARY_PATH"

# Set the variables for the platform release
# These are updated by the deploy script
export PEEK_ENV=""
[ -n "${PEEK_ENV}" ] && export PATH="${PEEK_ENV}/bin:$PATH"
```

Warning: Restart your terminal to get the new environment.

4.4.10 Compile and Install Python 3.9.1

The Peek Platform runs on Python. These instructions download, compile and install the latest version of Python.

Download and unarchive the supported version of Python:

```
cd
source .bashrc
wget https://github.com/python/cpython/archive/v${PEEK_PY_VER}.zip
unzip v${PEEK_PY_VER}.zip
cd cpython-${PEEK_PY_VER}
```

Configure the build:

```
./configure --prefix=/home/peek/opt/ --enable-optimizations --enable-shared
```

Make and Make install the software:

```
make install
```

Cleanup the download and build dir:

```
cd
rm -rf cpython-${PEEK_PY_VER}
rm v${PEEK_PY_VER}.zip
```

Symlink the python3 commands so they are the only ones picked up by path:

```
cd /home/peek/opt/bin
ln -s pip3 pip
ln -s python3 python
cd
```

Test that the setup is working:

```
RED='\033[0;31m'
GREEN='\033[0;32m'
NC='\033[0m' # No Color

SHOULD_BE="/home/peek/opt/bin/python"
if [ `which python` == ${SHOULD_BE} ]
then
    echo -e "${GREEN}SUCCESS${NC} The python path is right"
else
    echo -e "${RED}FAIL${NC} The python path is wrong, It should be ${SHOULD_BE}"
fi

SHOULD_BE="/home/peek/opt/bin/pip"
if [ `which pip` == ${SHOULD_BE} ]
then
    echo -e "${GREEN}SUCCESS${NC} The pip path is right"
else
    echo -e "${RED}FAIL${NC} The pip path is wrong, It should be ${SHOULD_BE}"
fi
```

Upgrade pip:

```
pip install --upgrade pip
```

synerty-peek is deployed into python virtual environments. Install the virtualenv python package:

```
pip install virtualenv
```

The Wheel package is required for building platform and plugin releases:

```
pip install wheel
```

4.4.11 Install PostgreSQL

Install the relational database Peek stores its data in. This database is PostgreSQL 12.

Note: Run the commands in this step as the peek user.

Download the PostgreSQL source code

```
PEEK_PG_VER=12.5
SRC_DIR="$HOME/postgresql-${PEEK_PG_VER}"

# Remove the src dir and install file
rm -rf ${SRC_DIR} || true
cd $HOME

wget https://ftp.postgresql.org/pub/source/v${PEEK_PG_VER}/postgresql-${PEEK_PG_VER}.tar.bz2
tar xjf postgresql-${PEEK_PG_VER}.tar.bz2

cd ${SRC_DIR}
```

Configure and build PostgreSQL

```
export CPPFLAGS="-I`echo $HOME/opt/include/python*m`"
export LDFLAGS="-L$HOME/opt/lib"

./configure \
    --disable-debug \
    --prefix=$HOME/opt \
    --enable-thread-safety \
    --with-openssl \
    --with-python

make -j4

make install-world

# this is required for timescale to compile
cp ${SRC_DIR}/src/test/isolation/pg_isolation_regress ~/opt/bin
```

Remove install files to clean up the home directory

```
# Remove the src dir and install file
cd
rm -rf ${SRC_DIR}*
```

Initialise a PostgreSQL database

```
#Refresh .bashrc so initdb can find postgres
source .bashrc

initdb --pgdata=$HOME/pgdata/12 --auth-local=trust --auth-host=md5
```

Tune the postgresql.conf

```
F="$HOME/pgdata/12/postgresql.conf"

sed -i 's/max_connections = 100/max_connections = 200/g' $F
```

Make PostgreSQL a service :

Note: This will require sudo permissions

Run the following command

```
touch postgresql-12.service

F=postgresql-12.service

cat <<"EOF" | sed "s,\$HOME,`echo ~peek`,g" > $F
[Unit]
Description=PostgreSQL 12 database server
After=syslog.target
After=network.target

[Service]
Type=forking
User=peek
Group=peek

# Location of database directory
Environment=PGDATA=$HOME/pgdata/12

# Disable OOM kill on the postmaster
OOMScoreAdjust=-1000
Environment=PG_OOM_ADJUST_FILE=/proc/self/oom_score_adj
Environment=PG_OOM_ADJUST_VALUE=0

ExecStart=$HOME/opt/bin/pg_ctl -D ${PGDATA} start
ExecStop=$HOME/opt/bin/pg_ctl -D ${PGDATA} stop
ExecReload=/bin/kill -HUP $MAINPID
KillMode=mixed
KillSignal=SIGINT

# Do not set any timeout value, so that systemd will not kill postmaster
# during crash recovery.
TimeoutSec=0

[Install]
WantedBy=multi-user.target
EOF

sudo mv $F /etc/systemd/system/postgresql-12.service
```

Reload the daemon

```
sudo systemctl daemon-reload
```

Install CMake

Download CMake source code:

```
PEEK_CMAKE_VER=3.19.2
SRC_DIR="$HOME/CMake-${PEEK_CMAKE_VER}"
wget https://github.com/Kitware/CMake/archive/v${PEEK_CMAKE_VER}.zip

unzip v${PEEK_CMAKE_VER}.zip
cd ${SRC_DIR}
```

Compile CMake from source:

```
./configure --prefix=$HOME/opt
make -j6 install

# Remove the src dir and install file
cd
rm -rf ${SRC_DIR}*
rm v${PEEK_CMAKE_VER}.zip
```

Install PostgreSQL Timescaledb

Next install timescaledb, this provides support for storing large amounts of historical data.

www.timescale.com

Download the timescaledb source code

```
PEEK_TSDB_VER=1.7.4

cd
wget https://github.com/timescale/timescaledb/archive/${PEEK_TSDB_VER}.zip
unzip ${PEEK_TSDB_VER}.zip
cd timescaledb-${PEEK_TSDB_VER}
```

Install the packages:

```
export CPPFLAGS=`pg_config --cppflags`
export LDFLAGS=`pg_config --ldflags`

# Bootstrap the build system
./bootstrap -DAPACHE_ONLY=1

# To build the extension
cd build && make

# To install
make install
```

(continues on next page)

(continued from previous page)

```
# Cleanup the source code
cd
rm -rf ${PEEK_TSDB_VER}.zip
rm -rf timescaledb-${PEEK_TSDB_VER}
```

Add the timescale repository:

```
curl -s https://packagecloud.io/install/repositories/timescale/timescaledb/script.deb.
↪sh | sudo bash
```

Install timescaledb-tune:

```
sudo apt install -y timescaledb-tools
```

Tune the database:

```
PGVER=12
FILE="$HOME/pgdata/${PGVER}/postgresql.conf"
timescaledb-tune -quiet -yes -conf-path ${FILE} -pg-version ${PGVER}
```

Start PostgreSQL:

```
systemctl enable postgresql-12 --now
```

Finish PostgreSQL Setup

Finish configuring and starting PostgreSQL.

Allow the peek OS user to login to the database as user peek with no password

```
F=$HOME/pgdata/12/pg_hba.conf
cat | sudo tee $F <<EOF
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all                                peek                    trust

# "local" is for Unix domain socket connections only
local   all                                all                      peer
# IPv4 local connections:
host    all                                all                      127.0.0.1/32            md5
# IPv6 local connections:
host    all                                all                      ::1/128                  md5
EOF
```

Create the database:

```
createdb -O peek peek
```

Set the PostgreSQL peek users password:

```
psql -d peek -U peek <<EOF
\password
\q
EOF

# Set the password as "PASSWORD" for development machines
# Set it to a secure password from https://xkpasswd.net/s/ for production
```

Note:

If you already have a database, you may now need to upgrade the `timescale` extension.

```
psql peek <<EOF ALTER EXTENSION timescaledb UPDATE; EOF
```

Cleanup traces of the password:

```
[ ! -e ~/.psql_history ] || rm ~/.psql_history
```

Grant PostgreSQL Peek Permissions

The PostgreSQL server now runs parts of peeks python code inside the postgres/postmaster processes. To do this the postgres user needs access to peeks home directory where the peek software is installed.

Grant permissions

```
sudo chmod g+rx ~peek
```

4.4.12 Install Worker Dependencies

Install the parallel processing queue we use for the peek-worker-service tasks.

Note: Run the commands in this step as the peek user.

Install redis and rabbitmq

```
sudo apt-get install -y redis-server rabbitmq-server
sudo apt-get clean
```

Enable the RabbitMQ management plugins:

```
sudo rabbitmq-plugins enable rabbitmq_mqtt
sudo rabbitmq-plugins enable rabbitmq_management
sudo service rabbitmq-server restart
```

Increase the size of the redis client queue

```
BEFORE="client-output-buffer-limit pubsub 64mb 16mb 90"
AFTER="client-output-buffer-limit pubsub 32mb 8mb 60"
sudo sed -i "s/{BEFORE}/{AFTER}/g" /etc/redis/redis.conf
sudo systemctl restart redis
```

4.4.13 Install Oracle Client (Optional)

The oracle libraries are optional. Install them where the agent runs if you are going to interface with an oracle database.

Warning: Open `~/.bashrc` insert the following at the start, before:

```
# If not running interactively, don't do anything
```

If you do not place the below code before that line, it will not be parsed.

```
# Setup the variables for ORACLE
export LD_LIBRARY_PATH="/home/peek/oracle/instantclient_21_1:$LD_LIBRARY_PATH"
export ORACLE_HOME="/home/peek/oracle/instantclient_21_1"
```

Source the new profile to get the new variables:

```
source ~/.bashrc
```

Make the directory where the oracle client will live

```
mkdir /home/peek/oracle
```

Download the following from oracle.

The version used in these instructions is **21.1.0.0.0**.

1. Download the ZIP “Basic Package” `instantclient-basic-linux.x64-21.1.0.0.0.zip` from <http://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html>
2. Download the ZIP “SDK Package” `instantclient-sdk-linux.x64-21.1.0.0.0.zip` from <http://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html>

Copy these files to `/home/peek/oracle` on the peek server.

Extract the files.


```
cd ~/oracle
unzip instantclient-basic-linux.x64-21.1.0.0.0.zip*
unzip instantclient-sdk-linux.x64-21.1.0.0.0.zip*
```

4.4.14 Install FreeTDS (Optional)

FreeTDS is an open source driver for the TDS protocol, this is the protocol used to talk to a MSSQL SQLServer database.

Peek needs this installed if it uses the pymssql python database driver, which depends on FreeTDS.

Warning: Open `~/ .bashrc` insert the following at the start, before:

```
# If not running interactively, don't do anything
```

If you do not place the below code before that line, it will not be parsed.

```
# Setup the variables for FREE TDS
export LD_LIBRARY_PATH="/home/peek/freetds:$LD_LIBRARY_PATH"
```

Warning: Restart your terminal you get the new environment.

Install FreeTDS:

```
sudo apt-get install -y freetds-dev
```

Create file `freetds.conf` in `~/freetds` and populate with the following:

```
mkdir ~/freetds
cat > ~/freetds/freetds.conf <<EOF

[global]
    port = 1433
    instance = peek
    tds version = 7.4

EOF
```

If you want to get more debug information, add the dump file line to the `[global]` section Keep in mind that the dump file takes a lot of space.

```
[global]
    port = 1433
    instance = peek
    tds version = 7.4
    dump file = /tmp/freetds.log
```

4.4.15 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

4.5 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

Deploy Peek Release

Note: The Windows or Debian requirements must be followed before following this guide.

This section describes how to deploy a peek platform release.

Peek is deployed into python virtual environments, a new virtual environment is created for every deployment.

This ensures that each install is clean, has the right dependencies and there is a rollback path (switch back to the old virtual environment).

To build your own platform release, see the following document *Package Peek Platform*.

5.1 Windows

5.1.1 Deploy Virtual Environment

Open a PowerShell window.

Download the platform deploy script. This is the only step in this section that requires the internet.

```
$deployScript = "deploy_release_win.ps1"
$uri = "https://bitbucket.org/synerty/synerty-peek/raw/master/scripts/win/
↪$deployScript";
[Net.ServicePointManager]::SecurityProtocol = "tls12, tls11, tls";
Invoke-WebRequest -Uri $uri -UseBasicParsing -OutFile $deployScript;
```

Run the platform deploy script. The script will complete with a print out of where the new environment was deployed.

Ensure you update the **\$platformArchive** variable with the path to your release.

The script will deploy to C:\Users\peek.

Tip: There are 80,000 files in the release, to speed up the extract, try these:

- Turn off antivirus, including the built in “Windows defender” in Win10
 - Ensure 7zip is installed, the deploy script checks and uses this if it’s present.
-

```
$platformArchive = "C:\Users\peek\Downloads\peek_platform_win_#.#.#.zip"
$pluginsArchive = "C:\Users\peek\Downloads\peek_plugins_win_#.#.#.zip"
PowerShell.exe -ExecutionPolicy Bypass -File $deployScript $platformArchive
↪ $pluginsArchive
```

When the script completes, you will be prompted to update the environment.

Press **Enter** to make this reality.

Otherwise, you will be given commands to temporarily configure the environment to use the synerty-peek virtual environment that was just deployed.

Now check that peek has been installed correctly, open a windows powershell prompt and enter the following:

```
get-command pip
# Expect to see C:\Users\peek\synerty-peek-1.0.0\Scripts\pip.exe

get-command python
# Expect to see C:\Users\peek\synerty-peek-1.0.0\Scripts\python.exe

python -c "import peek_platform; print(peek_platform.__file__)"
# Expect to see C:\Users\peek\synerty-peek-1.0.0\lib\site-packages\peek_platform\__
↪ init__.py
```

Note: If the paths are not as expected, ensure that the SYSTEM environment PATH variable does not contain any paths with “C:\Users\Peek\Python36” in it.

When a command prompt is open the order of PATH is SYSTEM then USER.

Peek on windows can run as a service. The following instructions are required to grant the “.peek” user permissions to start services (Grant “Login as Service”).

1. Run “services.msc”
2. Find the peek logic service
3. Open the properties of the service
4. Goto the LogOn tab
5. Enter the password twice and hit OK
6. A dialog box will appear saying that the Peek users has been granted the right.

Thats it, Peek can now start services.

The platform is now deployed, see the admin page next.

- *Configuring Platform config.json*
- *Run Peek Manually*

5.2 Linux

Run all commands from a terminal window remotely via ssh.

Download the platform deploy script.

Note: This is the only step in this section that requires the internet. If you don't have internet access you may try this command, be sure to update the "servername" to the server ip address: `scp Downloads/deploy_release_linux.sh peek@servername:/home/peek/deploy_release_linux.sh`

```
uri="https://bitbucket.org/synerty/synerty-peek/raw/master/scripts/linux/deploy_
↪release_linux.sh"
wget $uri
```

Run the platform deploy script. The script will complete with a print out of where the new environment was deployed.

Ensure you update the **dist** variable with the path to your release.

The script will deploy to `/home/peek/`.

```
communityArchive="/home/peek/Downloads/peek_community_linux_#.#.#.tar.bz2"
enterpriseArchive="/home/peek/Downloads/peek_enterprise_linux_#.#.#.tar.bz2"
bash deploy_release_linux.sh $communityArchive $enterpriseArchive
```

Once the script has completed running you will see the message "Activate the new environment edit ...".

This command configures the environment to use the synerty-peek virtual environment that was just deployed.

The platform is now deployed, see the admin page next.

- *Configuring Platform config.json*
- *Run Peek Manually*

5.3 macOS

Run all commands from a terminal window remotely via ssh.

Download the platform deploy script.

Note: This is the only step in this section that requires the internet. If you don't have internet access you may try this command, be sure to update the "servername" to the server ip address: `scp Downloads/deploy_release_macos.sh peek@servername:/Users/peek/deploy_release_macos.sh`

```
file="deploy_release_macos.sh"
uri="https://bitbucket.org/synerty/synerty-peek/raw/master/scripts/macOS/$file"
wget $uri
```

Run the platform deploy script. The script will complete with a print out of where the new environment was deployed.

Ensure you update the **dist** variable with the path to your release.

The script will deploy to `/Users/peek/`.

```
communityArchive="/home/peek/Downloads/peek_community_macos_#.#.#.tar.bz2"
enterpriseArchive="/home/peek/Downloads/peek_enterprise_macos_#.#.#.tar.bz2"
bash deploy_release_macos.sh $communityArchive $communityArchive
```

Once the script has completed running you will see the message "Activate the new environment edit ...".

This command configures the environment to use the synerty-peek virtual environment that was just deployed.

The platform is now deployed, see the admin page next.

- [Configuring Platform config.json](#)
- [Run Peek Manually](#)

5.4 Development Considerations

Deploying an new platform will clear out some of the setup for developing plugins or the platform.

If you've run these commands as part of any development setups, you'll need to run them again now

Example, run this for each python package/plugin you're developing.

```
python setup.py develop
```

5.5 What Next?

Refer back to the [How to Use Peek Documentation](#) guide to see which document to follow next.

This document set provides administration documentation for the Peek Platform.

Administration documentation for plugins loaded in to the platform can be viewed on the Peek server at <http://127.0.0.1:8015>.

6.1 Configuring Platform `config.json`

Update `config.json` files. This tells the peek platform services how to connect to each other, connect to the database, which plugins to load, etc.

Note: Running the services of Peek will automatically create and fill out the missing parts of `config.json` files with defaults. So we can start with just what we want to fill out.

6.1.1 Peek Logic Service

This section sets up the config files for the peek **logic** service.

Create following file and parent directory:

Windows `C:\Users\peek\peek-logic-service.home\config.json`

Linux `/home/peek/peek-logic-service.home/config.json`

Mac `/Users/peek/peek-logic-service.home/config.json`

Tip: Run the service, it will create some of it's config before failing to connect to the db.

Populate the file `config.json` with the

- SQLAlchemy connect URL (See options below)
- Enabled plugins

Select the right `connectUrl` for your database, ensure you update `PASSWORD`.

MS Sql Server `mssql+pymssql://peek:PASSWORD@127.0.0.1/peek`

PostgreSQL `postgresql://peek:PASSWORD@127.0.0.1/peek`

```
{
  "plugin": {
    "enabled": [
      "peek_plugin_inbox",
      "peek_plugin_tutorial"
    ]
  },
  "sqlalchemy": {
    "connectUrl": "postgresql://peek:PASSWORD@127.0.0.1/peek"
  }
}
```

6.1.2 Peek Field Service

This section sets up the config files for the peek **field** service.

Create following file and parent directory:

Windows `C:\Users\peek\peek-field-service.home\config.json`

Linux `/home/peek/peek-field-service.home/config.json`

Mac `/Users/peek/peek-field-service.home/config.json`

Tip: Run the service, it will create some of it's config, it might raise errors though.

Populate the file `config.json` with the

- Enabled plugins

```
{
  "plugin": {
    "enabled": [
      "peek_plugin_inbox",
      "peek_plugin_tutorial"
    ]
  }
}
```


6.1.3 Peek Office Service

This section sets up the config files for the peek **office** service.

Create following file and parent directory:

Windows C:\Users\peek\peek-office-service.home\config.json

Linux /home/peek/peek-office-service.home/config.json

Mac /Users/peek/peek-office-service.home/config.json

Tip: Run the service, it will create some of it's config, it might raise errors though.

Populate the file `config.json` with the

- Enabled plugins

```
{
  "plugin": {
    "enabled": [
      "peek_plugin_inbox",
      "peek_plugin_tutorial"
    ]
  }
}
```

6.1.4 Peek Agent Service

This section sets up the config files for the peek **agent** service.

Create following file and parent directory:

Windows C:\Users\peek\peek-agent-service.home\config.json

Linux /home/peek/peek-agent-service.home/config.json

Mac /Users/peek/peek-agent-service.home/config.json

Tip: Run the service, it will create some of it's config, it might raise errors though.

Populate the file `config.json` with the

- Enabled plugins

```
{
  "plugin": {
    "enabled": [
      "peek_plugin_inbox",
      "peek_plugin_tutorial"
    ]
  }
}
```

(continues on next page)

(continued from previous page)

```
    ]  
  }  
}
```

6.1.5 Peek Field, Office, Logic Service SSL

This section sets up SSL for the peek field, office and logic services.

Combine the required SSL certificates and keys into a single PEM file named `peek-ssl-bundle.pem`.

For example, this can be done on Linux by concatenating the Key, Cert and CA files.

```
cat key.pem cert.pem ca.pem > bundle.pem
```

Note: The file names will vary, but the file contents will start with lines like the following

```
==> CA cert <==
```

—BEGIN CERTIFICATE—

```
==> Cert <== —BEGIN CERTIFICATE—
```

```
==> Key <== —BEGIN RSA PRIVATE KEY—
```

Place a copy of this PEM file into the server directory:

Windows `C:\Users\peek\peek-logic-service.server\peek-ssl-bundle.pem`

Linux `/home/peek/peek-logic-service.home/peek-ssl-bundle.pem`

Mac `/Users/peek/peek-logic-service.home/peek-ssl-bundle.pem`

Restart the Peek server service.

Place a copy of this PEM file into the field directory:

Windows `C:\Users\peek\peek-field-service.server\peek-ssl-bundle.pem`

Linux `/home/peek/peek-field-service.home/peek-ssl-bundle.pem`

Mac `/Users/peek/peek-field-service.home/peek-ssl-bundle.pem`

Place a copy of this PEM file into the office directory:

Windows `C:\Users\peek\peek-office-service.server\peek-ssl-bundle.pem`

Linux `/home/peek/peek-office-service.home/peek-ssl-bundle.pem`

Mac `/Users/peek/peek-office-service.home/peek-ssl-bundle.pem`

Restart the Peek field and office services.

The Peek logic service, field service, and office service should now be using SSL.

6.2 Run Peek Manually

This section describes the best practices for running the peek platform manually

To use bash on windows, install msys git. *Setup Msys Git*, otherwise use powershell on windows.

6.2.1 Check Environment

Make sure that the right environment is activated. Run the following commands.

PowerShell

```
(Get-Command python).source  
(Get-Command run_peek_logic_service).source
```

Or Bash

```
which python  
which run_peek_logic_service
```

Confirm that the output contains the release you wish to use.

6.2.2 run_peek_logic_service

This section runs the peek server service of the platform and opens the admin page.

Run the following in bash, cmd or powershell

```
run_peek_logic_service
```

Open the following URL in a browser, Chrome is recommended.

<http://127.0.0.1:8010/>

This is the administration page for the peek platform, otherwise known as the “Admin” service.

6.2.3 run_peek_office_service

This section runs the peek client service, this serves the desktop and mobile web apps and provides data to all desktop and mobile native apps

Run the following in bash, cmd or powershell

```
run_peek_office_service
```

Open the following URL in a browser, Chrome is recommended.

<http://127.0.0.1:8000/>

This is the mobile web app for the peek platform.

6.2.4 run_peek_agent_service

The Agent is used to connect to external systems, this section runs the agent service.

Run the following in bash, cmd or powershell

```
run_peek_agent_service
```

6.2.5 Whats Next

Now that the platform is running, See the next section, *Updating Plugin Settings*

6.3 Logs

This document provides information on the Peek Platform logs.

The Peek Platform has several services, each one of these services has their own log.

The logs are written to the peek user's home directory, for example, the Peek Logic Service logfile is located at `/home/peek/peek-logic-service.log`.

The logs are rotated when they reach 20mb, maintaining the last two old log files.

The log level for each service can be configured in the `services config.json` located in the services config directory for example `/home/peek/peek-logic-service.home/config.json`.

Change the log level with this setting:

```
"logging": {
  "level": "DEBUG"
},
```

The value must match the “Level” column in this document : <https://docs.python.org/3.6/library/logging.html?highlight=logging#levels>

System Administrators should monitor the Peek logs. Generally only WARNING or ERROR messages will require some concern.

6.4 Getting Support

Peek is an open source platform, Synerty provides a best effort response for community requested support, and we welcome good pull requests

6.4.1 Enterprise Support

Synerty offers Enterprise support for enterprise customers,

Enterprise customers receive the following support as part of their support package.

1. 24/7 Phone Support.
2. Instant message support.
3. Email Support.
4. Screen sharing support.
5. Issue management system access.

6.4.2 Reporting Bugs

Reporting bugs with a good level of detail helps Synerty Developers quickly identify the cause of the problem and provide a fix.

The following information should be included in any bugs submitted to Synerty.

Summary of Issue A short title that allows this issue to be recognised amongst a list of many other issues.
Try including a distinct detail of this issue.

Detailed description of bug A full scription of the issue, including:

- The versions of the peek packages `pip freeze`
- The list of enabled plugins for the effected Peek service.
- The Peek server operating system type
- What is the observed behavior
- What is the intended behavior
- Steps to Reproduce the issue
- Any other information, such as recent changes to the system, etc.

Attach logs Attach zipped logs from the Peek servers, or a extract from the logs.

Attach screen shots Screenshots are really helpful.

Do include screen shots of screens, etc.

Don't include screenshots of terminals, logs, etc, copy the text from the terminal instead.

Don't attach screenshots as word documents, or zipped up word documents.

Do attach images directly to issues, or inserted inline with email content.

Bugs can be composed in word documents, email contents and then submitted to Synerty, or submitted directly to Synerty issue management system.

6.5 Updating Plugin Settings

Plugins are intended to be entirely configured via the peek server Admin page.

Navigate to <http://127.0.0.1:8010/> and click the plugins dropdown.

6.6 Peek Windows Admin

6.6.1 Windows Services

You only need to start “peek-logic-service” and “peek-restarter”.

If you want to restart peek, just restart “peek-logic-service”, the worker, agent, field, and office services will shutdown and be restarted.

The **peek-restarter** service automatically restarts the **worker**, **field**, **office**, and **agent** services. On windows, these services will stop when the **peek-logic-service** stops.

6.6.2 Backup and Restore PostgreSQL DB

Backup

This section describes how to backup the PostgreSQL database for Peek on a windows server.

Open a Powershell window, and change directory to the location of where you want the backup placed.

For example:

```
cd 'C:\Users\Peek\Backups\'
```

Run the following command to execute the backup.

This will create a plain text SQL backup of the database.

```
pg_dump -h 127.0.0.1 -U peek -d peek -F p -f peek_backup.sql
```

Here is another example that provides a smaller backup.

The bulk of the data is left behind, and the loader state tables are reset so the data is reloaded when the destination peek starts.

```
pg_dump -h 127.0.0.1 -U peek -d peek -F p -f peek_backup.sql `
--exclude-table-data 'pl_diagram.\"DispBase\"' `
--exclude-table-data 'pl_diagram.\"DispEllipse\"' `
--exclude-table-data 'pl_diagram.\"DispGroup\"' `
--exclude-table-data 'pl_diagram.\"DispGroupItem\"' `
--exclude-table-data 'pl_diagram.\"DispGroupPointer\"' `
--exclude-table-data 'pl_diagram.\"DispPolygon\"' `
--exclude-table-data 'pl_diagram.\"DispPolyline\"' `
--exclude-table-data 'pl_diagram.\"DispText\"' `
--exclude-table-data 'pl_diagram.\"LiveDbDispLink\"' `
```

(continues on next page)

(continued from previous page)

```
--exclude-table-data 'pl_diagram.\"DispCompilerQueue\"' `
--exclude-table-data 'pl_diagram.\"GridKeyIndex\"' `
--exclude-table-data 'pl_diagram.\"GridKeyCompilerQueue\"' `
--exclude-table-data 'pl_diagram.\"GridKeyIndexCompiled\"' `
--exclude-table-data 'pl_diagram.\"LocationIndex\"' `
--exclude-table-data 'pl_diagram.\"LocationIndexCompilerQueue\"' `
--exclude-table-data 'pl_diagram.\"LocationIndexCompiled\"' `
--exclude-table-data 'pl_diagram.\"BranchIndex\"' `
--exclude-table-data 'pl_diagram.\"BranchIndexEncodedChunk\"' `
--exclude-table-data 'pl_diagram.\"BranchIndexCompilerQueue\"' `
--exclude-table-data 'pl_docdb.\"DocDbDocument\"' `
--exclude-table-data 'pl_docdb.\"DocDbChunkQueue\"' `
--exclude-table-data 'pl_docdb.\"DocDbEncodedChunkTuple\"' `
--exclude-table-data 'core_search.\"SearchIndex\"' `
--exclude-table-data 'core_search.\"SearchIndexCompilerQueue\"' `
--exclude-table-data 'core_search.\"EncodedSearchIndexChunk\"' `
--exclude-table-data 'core_search.\"SearchObject\"' `
--exclude-table-data 'core_search.\"SearchObjectRoute\"' `
--exclude-table-data 'core_search.\"SearchObjectCompilerQueue\"' `
--exclude-table-data 'core_search.\"EncodedSearchObjectChunk\"' `
--exclude-table-data 'pl_branch.\"BranchDetail\"' `
--exclude-table-data 'pl_livedb.\"LiveDbItem\"' `
--exclude-table-data 'pl_graphdb.\"GraphDbChunkQueue\"' `
--exclude-table-data 'pl_graphdb.\"GraphDbEncodedChunk\"' `
--exclude-table-data 'pl_graphdb.\"GraphDbSegment\"' `
--exclude-table-data 'pl_graphdb.\"ItemKeyIndex\"' `
--exclude-table-data 'pl_graphdb.\"ItemKeyIndexCompilerQueue\"' `
--exclude-table-data 'pl_graphdb.\"ItemKeyIndexEncodedChunk\"' `
--exclude-table-data 'pl_enmac_user_loader.\"LoadState\"' `
--exclude-table-data 'pl_gis_diagram_loader.\"DxfLoadState\"' `
--exclude-table-data 'pl_enmac_gis_location_loader.\"ChunkLoadState\"' `
--exclude-table-data 'pl_enmac_diagram_loader.\"PageLoadState\"' `
--exclude-table-data 'pl_enmac_graphdb_loader.\"GraphSegmentLoadState\"' `
--exclude-table-data 'pl_enmac_switching_loader.\"ChunkLoadState\"' `
--exclude-table-data 'pl_enmac_equipment_loader.\"ChunkLoadState\"'
```

OR, This will create a more binary backup format, suitable for restoring onto an existing peek server. Some databases modules such as postgis, etc will not be dumped with the custom format.

To backup to the custom format change `-F p` to `-F c` and change the file name extension from `.sql` to `.dmp`.

```
pg_dump -h 127.0.0.1 -U peek -d peek -F c -f peek_backup.dmp `
```

Restore

This section describes how to restore the PostgreSQL database for Peek on a windows server.

Warning: This procedure deletes the existing Peek database. Ensure you have everything in order, backed up and correct before executing each command. (Including the server your connected to)

Stop all Peek services from the windows services.

These can be quickly accessed by pressing CTRL+ESC to bring up the task manager and then selecting the services tab.

Look in the windows tray / notifications area to see if the **PGAdmin4** server is running.

If it is, right click on it and select **Shutdown Server**

Open a Powershell window, and change directory to the location of the backup. For example:

```
cd 'C:\Users\Peek\Downloads\v1.1.6.3\'
```

Run the command to drop the existing Peek database. You won't see any errors or feedback when this succeeds.

```
dropdb -h 127.0.0.1 -U peek peek
```

Run the command to create a fresh new Peek database. You won't see any errors or feedback when this succeeds.

```
createdb -h 127.0.0.1 -U peek -O peek peek
```

To restore a Plain SQL backup (created with `-F p` and extension `.sql`) use this section.

Restore the PostgreSQL database. This will create the schema and load the data.

```
psql.exe -h 127.0.0.1 -U peek -d peek -f .\peek_backup.sql
```

OR, To restore a Custom backup (created with `-F c` and extension `.dmp`) use this section.

Restore the PostgreSQL database. This will create the schema and load the data.

```
pg_restore.exe -h 127.0.0.1 -U peek -d peek peek_backup.dmp
```

7.1 Build Capacitor App

7.1.1 Install Capacitor

Capacitor can be installed via NPM.

```
# Install capacitor
cd peek_field_app # or peek_office_app
npm install @capacitor/core @capacitor/cli --save

# Initialise capacitor
npx cap init

# Setup iOS development
npx cap add ios
```

7.1.2 Build iOS App

Capacitor iOS apps can be built either via the xcode interface, or by using the xcodebuild cli. This document will cover building the app using the xcodebuild cli.

```
# Build iOS archive
xcodebuild -workspace App.xcworkspace -scheme App archive -archivePath build/peek.
↳ xcarchive

# Create iOS app from archive
xcodebuild -exportArchive -archivePath build/peek.xcarchive -exportPath release/Peek -
↳ exportOptionsPlist OptionsPlist.plist
```

7.1.3 Build Android App

** ToDo **

7.1.4 Build Windows App

** ToDo **

7.1.5 Further Reading

Further documentation on Capacitor can be found on their website. <https://capacitorjs.com/docs>

7.1.6 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

7.2 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

8.1 Develop Peek Plugin Guides

8.1.1 Develop Peek Plugins

Synerty recommends the Atlassian suite of developer tools.

Bitbucket to manage and share your Git repositories

URL <https://www.bitbucket.org>

SourceTree to visually manage and interact with your Git repositories

URL <https://www.sourcetreeapp.com>

Bitbucket can be integrated with Jira (issue management) and Bamboo (continuous integration).

Note: The reader needs be familiar with, or will become familiar with the following:

- [GIT](#)
- [Python3.5+](#)
- [Python Twisted](#)
- [HTML](#)
- [SCSS](#)
- [Ant Design](#)
- [TypeScript](#)
- [Angular](#) (Angular2+, not AngularJS aka Angular1)
- [Capacitor](#)

Note: This a cross platform development guide, all commands are written for bash.

Bash is installed by default on Linux.

Windows users should use bash from msys, which comes with git for windows, *Setup Msys Git*.

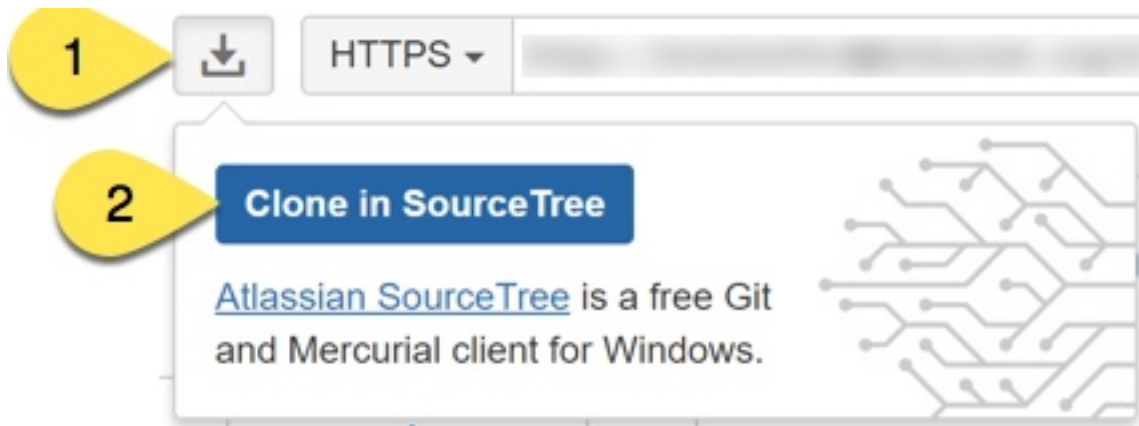
Clone a New Peek Plugin

If you're creating a new plugin you can copy from "peek-plugin-noop" and rename.

Copy peek-plugin-noop

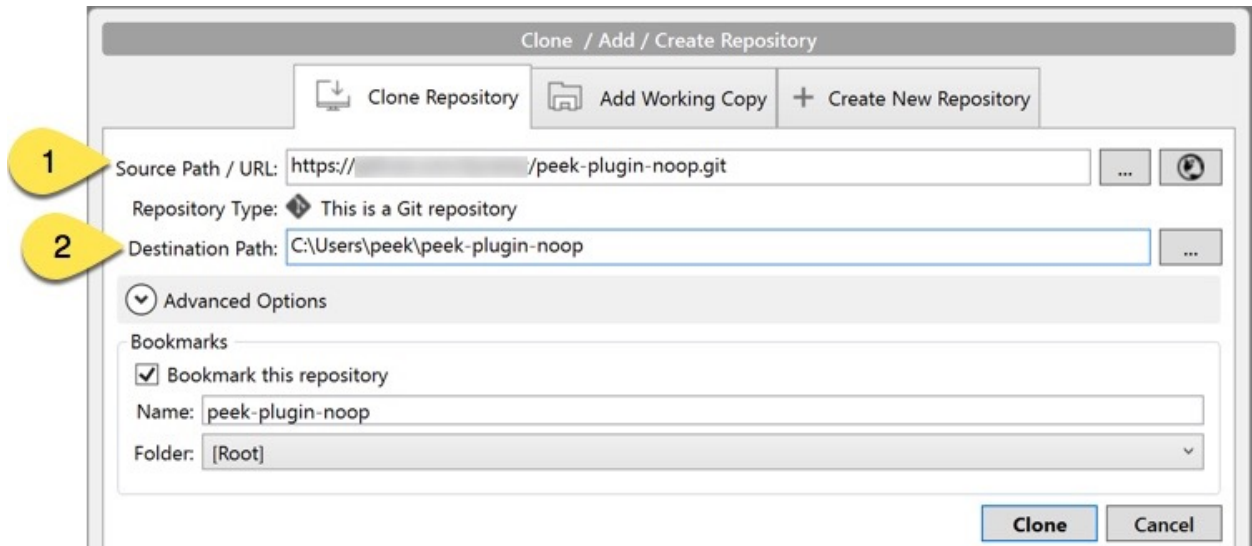
Clone <https://bitbucket.org/synerty/peek-plugin-noop/src>

Go to, peek-plugin-noop repository on Bitbucket



Clone the repository

1. This URL will be automatically populated from Bitbucket.
2. **Alter this name to end with peek-plugin-example.**



Remove the git references into new directory structure, run the following commands in the bash shell:

```
cd peek-plugin-example
rm -rf .git .idea .vscode
```

Rename to New Plugin

Edit the `rename_plugin.sh` file in the plugin root project folder.

Update the variables near the top with the new names:

```
caps="EXAMPLE"
underscore="_example"
hyphen="-example"
camelL="example"
camelU="Example"
```

Run `rename_plugin.sh`, run the following command in the bash shell:

```
bash ./rename_plugin.sh
```

Remove the “`rename_plugin.sh`” script, run the following command in the bash shell:

```
rm rename_plugin.sh
```

Add to GIT

Create new repository on GitHub.

Create a new repository

[Import repository](#)

Repository name *

example

Access level

☒ This is a private repository

Repository type

☒ Git

☐ Mercurial

[Advanced settings](#)

Create repository

Cancel

Note: Bitbucket will also provide instructions on how to do the following.

Get the git url, it will look something like:

```
https://{account username}@bitbucket.org/{account username}/example.git
```

Run the following commands in bash shell to add the plugin to the git repository:

```
git init
git add .
```

Create your first commit:

```
git commit -m "Scaffolded example plugin"
```

Add remote:

```
git remote add origin {insert your GitHub link}
```

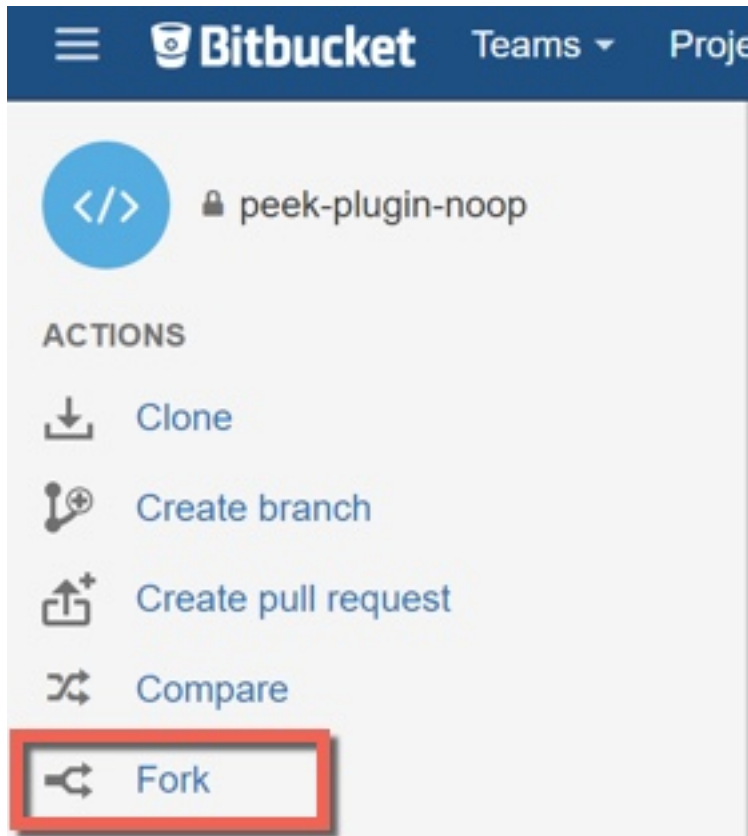
Push your changes:

```
git push -u origin master
```

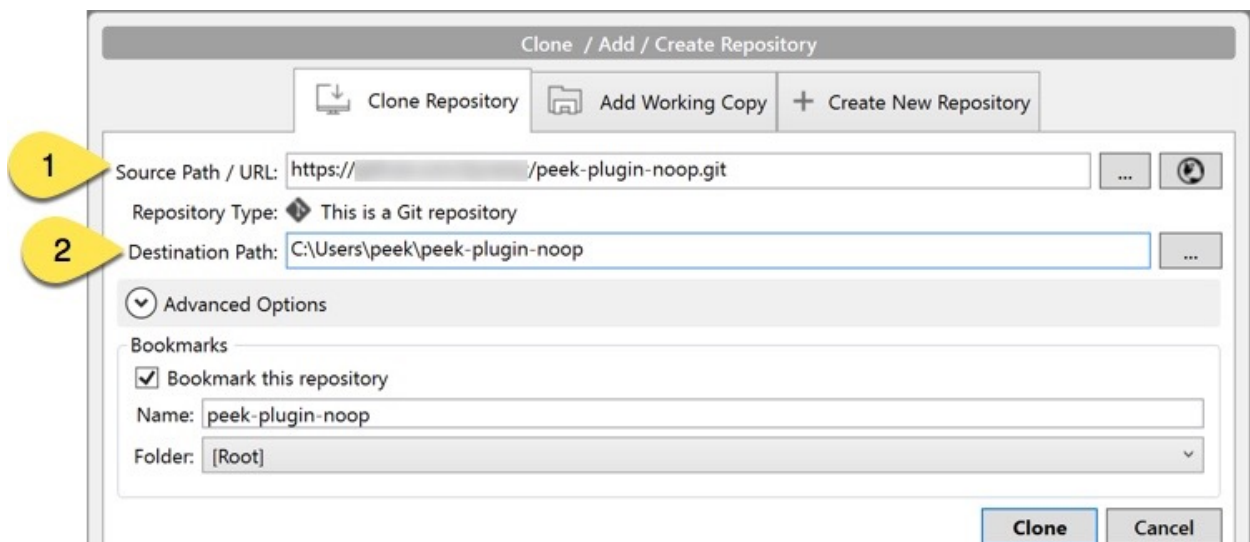
Cloning an Existing Peek Plugin

Create your own fork of the plugins if you don't already have one.

Warning: Be sure to check your fork syncing is enabled and up to date, Otherwise you'll run into issues.



Clone the fork



Setup an IDE

An integrated development environment (IDE), is an advanced text editor with the following features.

- Syntax highlighting
- Error highlighting
- Integrating build tools
- Debugging
- Linting - checking code for quality.

The Peek documentation has procedures for IDE setup:

- [Setup Pycharm IDE](#)
- [Setup VS Code IDE](#)

8.1.2 Setup Plugin for Development

Plugins need to be installed as python packages for the Peek Platform to run them. This is typically done with a command similar to **pip install peek-plugin-noop**.

Python packages can be installed in “development” mode, where your code being developed is only linked into the python environment.

Note: For developing an existing plugin ensure there are no installed releases `pip uninstall peek-plugin-example`. Confirm installed peek packages with `pip freeze | grep peek`.

This is achived with the following command in the plugin project root directory, where setup.py is:

```
# Check to ensure we're using the right python
which python

python setup.py develop
```

Configure Peek Services

The python peek services, **worker**, **agent**, **field**, **office**, and **logic** need to have the plugin enabled in their `~/peek-service/config.json`.

For example:

```
"plugin": {
  "enabled": [
    "peek_plugin_example"
  ]
}
```


Run the Plugin

Now that the plugin has been setup for development and the platform has been configured to run it, running the platform will run the plugin.

See the Setup IDE procedures to run the platform and debug plugins under those.

If a platform service, (**run_peek_logic_service** for example) is run under the IDEs debugger, it will also debug the plugins the platform loads.

Run the platform services from bash with the following commands:

```
# Check to ensure we're using the right python
which python

# Run the peek logic service
run_peek_logic_service

# Run the peek office service
run_peek_office_service

# Run the peek field service
run_peek_field_service

# Run the peek agent
run_peek_agent_service

# Run the peek worker
run_peek_worker_service
```

8.1.3 Developing With The Frontends

The Peek Platform is extensible with plugins. Unlike with the Python code, the frontend code can't be just imported. The frontend code in the plugins have to be combined into build directories for each service.

This document describes how Peek combines the Frontend / Angular files into build projects.

The frontends are the Admin, Mobile and Desktop services.

- Admin builds:
 - Only a Web app, using @angular/cli
- Field builds:
 - A Web app, using @angular/cli
- Office builds:
 - An Electron app.
 - A Web app, using @angular/cli

The platform code for combining the files for the frontends is at: https://bitbucket.org/synerty/peek-platform/src/master/peek_platform/build_frontend/

Combining Files

The Logic, Office, and Field services prepare the build directories for all the frontends.

Peek originally used symbolic links to integrate the plugins, this approach become harder and harder to manage with both cross platform support and increasing complexity of the plugin integrations with the frontend.

Peek now uses a file copy approach, that is handled by the Logic, Office, and Field services. There were many things to consider when implementing this code, considerations include:

Incremental file updates. Don't rewrite the file if it hasn't changed. This causes problems with development tools incorrectly detecting file changes.

Allow on the fly modifications. Peek rewrites parts of the frontend code on the fly.

Angular Ahead of Time Compilation. The final nail in the symlink approach was Angular AoT Compilation support.

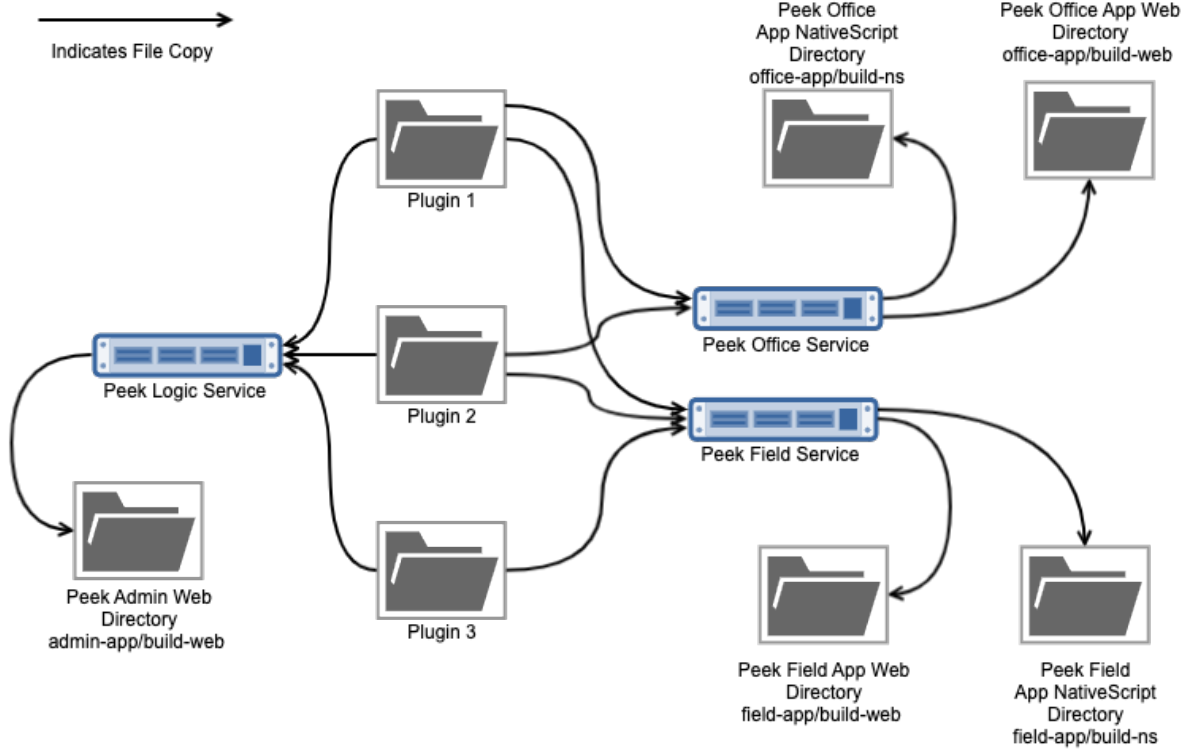
Plugins in node_modules. Plugins in the frontends can leverage each other. For example, other plugins can leverage services from `peek_core_user` for user authentication. To do this, Peek copies files into `node_modules`.

End user customisation. The ability for the end user to overwrite files with out having to fork plugins, etc.

The file copy process, AKA “prepare” or “Combine”, is fairly simple:

1. Use the list of loaded plugins
2. Read the `plugin_package.json`
3. Copy and transform files based on `plugin_package.json` settings.
4. Create static files for:
 1. Lazy loading routes for plugins
 2. Global services from plugins
 3. Home icon and Menu items
 4. etc
5. Compile plugin code copied to `node_modules`.
6. Copy required files to the platform directory.

At this point the build directories are prepared and ready to run.



End User Customisations

End users, as in not developers, have the ability to hack the frontends. They may do this to change text, update icons, branding, etc.

Note: The following directory overlay is for the web app directory.

Peek reads files from either the `~/peek-logic-service.home/frontendSrcOverlayDir` or `~/peek-office-service.home/frontendSrcOverlayDir` directories and overlays them on top of the build directories.

This provides end users with the ability to alter any part of the Electron, Web or Capacitor frontends by copying a file into the customisation directory and then altering it.

This is a use at their own risk feature.

The following property is present in the Peek Logic, Office, and Field `*-service.home/config.json` files.

```
{
  ...
  "frontend": {
    ...
    "frontendSrcOverlayDir": "/home/peek/peek-office-service.home/
    ↪frontendSrcOverlayDir",
    "frontendNodeModuleOverlayDir": "/home/peek/peek-office-service.home/
    ↪frontendNodeModuleOverlayDir",
  },
}
```

(continues on next page)

(continued from previous page)

```
}  
...  
}
```

#1 Copy the plugin build directory into the frontend folder

```
cp -r synerty-peek-1.3.4/lib/python3.6/site-packages/peek_office_app/src/peek_plugin_  
enmac_diagram peek-office-service.home/frontendSrcOverlayDir/
```

#2 Remove the files not being updated (we're updating pofDiagram.component.web.html)

```
cd peek-office-service.home/frontendSrcOverlayDir/peek_plugin_enmac_diagram/  
rm -rf coord-set/ show-diagram/ *.ts
```

#3 Edit the file and restart Peek

```
vi pofDiagram.component.web.html  
restart_peek.sh
```

#4 Monitor the logs and refresh Peek Desktop

```
http://peekserver:8002/peek_plugin_enmac_diagram
```

5 Undo the changes

```
rm -rf ~/peek-office-service.home/frontendSrcOverlayDir/peek_plugin_enmac_diagram  
restart_peek.sh
```

Live Updating for Development

Both **Capacitor** and **Angular CLI** have development tools that provide live sync + refresh support.

Meaning, you can alter your code, save, and the tools will recompile, and update the apps. Angular CLI will update the code for the web page and reload it, Capacitor will copy the generated bundles to the iOS and Android directories to be used in xcode and Android Studio.

Peek's frontend preparation code creates maps of where files should be copied from and to, then monitors all the source directories, and incrementally updates files as the developer works. This includes performing any on the fly changes to the files that are required.

To enable file syncing, in file(s) `~/peek-logic-service.home/config.json` or `~/peek-office-service.home/config.json` set `frontend.syncFilesForDebugEnabled` to `true` and restart the appropriate service.

You may also want to disable the web building. This isn't required for the Angular CLI development server and it slows down Peek Logic, Office, and Field restarts. Set `frontend.webBuildEnabled` to `false`.

If `DEBUG` logging is also enabled, you'll see Peek working away when you change files.

```
{  
  ...  
  "frontend": {  
    ...  
    "syncFilesForDebugEnabled": true,  
    "webBuildEnabled": false,  
  },  
}
```

(continues on next page)

(continued from previous page)

```

    ....
  },
  "logging": {
    "level": "DEBUG"
  },
  ...
}

```

Now when you run:

```

# Start Angular CLI live dev server
npm start

```

The web apps will automatically update as the developer changes things.

build-web

To build the dist dir, and serve it on a normal port run:

```
ng build -w
```

The `-w` option listens for changes.

To run the packages start scripts run:

```
npm start
```

Auto refreshes, deletes the dist that ng build creates, and the proxy settings for file resources and http vortex.

Developing on iOS Devices

Before Peek can be deployed the signing certificate must be transferred to the device using Xcode.

To develop with iOS you'll need a developer account on <https://developer.apple.com>

Troubleshooting

OSError: inotify instance limit reached

If you receive an error when starting the Peek Logic, Office, and Field services on Linux, stating `OSError: inotify instance limit reached`, running the following command may solve the issue.

```
sudo sysctl fs.inotify.max_user_watches=200000
```

Otherwise, try rebooting.

8.1.4 Continue Development

To learn more about plugin development from scratch, or the basic setup of plugins, see *Peek Plugin Tutorial*.

8.1.5 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

8.2 Publish Peek Plugins

The peek package has build scripts that generate a platform build.

Important: Windows users must use bash.

8.2.1 Create Private Plugin Release

Note: Do not follow this step if you intend on using a public release, see *Create PyPI Public Release*

Change root directory of peek-plugin, example:

```
cd peek-plugin-example/
```

Ensure RELEASE_DIR is where you want the release:

```
echo $RELEASE_DIR
```

Ensure that the file `publish.sh` variable `PYPI_PUBLISH` is blank

```
# Leave blank not to publish
# Or select one of the index servers defined in ~/.pypirc
PYPI_PUBLISH=""
```

Run the follow command being sure to increment the version number:

```
./publish.sh #.#.#
```

Expected response like:

```
$ ./publish.sh 0.0.7
Setting version to 0.0.7

...

Not publishing to any pypi indexes
```

8.2.2 Create PyPI Public Release

The Python Package Index is a repository of software for the Python programming language.

Setting up your PyPI Accounts

First you will need to create your user account.

Register here: [PyPI](#)

Create file `~/.pypirc`

Create file `~/.pypirc` and populate with the following:

```
[distutils]
index-servers=
    pypi

[pypi]
repository = https://pypi.python.org/pypi
username = <your user name goes here>
password = <your password goes here>
```

Note: Make sure you update the username and password.

Run script `publish.sh`

Change root directory of peek-plugin, example:

```
cd peek-plugin-example/
```

Ensure `RELEASE_DIR` is where you want the release:

```
echo $RELEASE_DIR
```

Ensure that the file `publish.sh` variable `PYPI_PUBLISH` is set to the index of the PyPI server defined in `~/.pypirc`:

```
# Leave blank not to publish
# Or select one of the index servers defined in ~/.pypirc
PYPI_PUBLISH="pypi"
```

Run the follow command, being sure to increment the version number:

```
./publish.sh #.#.#
```

Expected response like:

```
$ ./publish.sh 0.0.7
Setting version to 0.0.7
...
```

(continues on next page)

(continued from previous page)

```
Writing peek-plugin-tutorial-0.0.7\setup.cfg
Creating tar archive
removing 'peek-plugin-tutorial-0.0.7' (and everything under it)
running upload
Submitting dist\peek-plugin-tutorial-0.0.7.tar.gz to https://upload.pypi.org/legacy/
Server response (200): OK
```

Check uploaded release on [PyPI](#).

8.2.3 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

8.3 Package Peek Plugins

8.3.1 Packaging a Production Release

A release is a zip file containing all the required python packages to install the plugins after the platform release has installed.

Important: Windows users must use bash. *Setup Msys Git*

Create the release directory:

```
mkdir ~/plugin-release-dir
```

Note: You should clean up any previously packaged releases: `rm -rf ~/plugin-release-dir`

Change to release directory:

```
cd ~/plugin-release-dir
```

Copy your private plugins “source distributions” into the release directory.

OPTION 1)

To build a source distribution, cd to the plugin dir and run the following:

```
# build the source distribution
cd ~/project/peek-plugin-example
python setup.py sdist

# Copy the source distribution to our release dir
cp ~/project/peek-plugin-example/dist/peek-plugin-example-#.#.#.tar.gz ~/plugin-
↪release-dir
```

OPTION 2)

The documentation to create plugins includes a `publish.sh` script, this does the following:

- Checks for uncommitted changes
- Updates version numbers on various files in the code
- Commits the version updates
- Tags the commit
- Optionally, uploads the plugin to PYPI
- Optionally, copies the dist to **\$RELEASE_DIR**

```
export RELEASE_DIR=`ls -d ~/plugin-release-dir`

# build the source distribution
cd ~/project/peek-plugin-example
bash publish.sh #.#.#

# Where #.#.# is the new version
```

Note: Repeat this step for each private plugin.

Make a wheel dir for windows or Linux.

Windows:

```
mkdir ~/plugin-release-dir/plugin-win
cd ~/plugin-release-dir/plugin-win
```

Linux:

```
mkdir ~/plugin-release-dir/plugin-linux
cd ~/plugin-release-dir/plugin-linux
```

Build Wheel archives for your private requirements and dependencies. Wheel archives are “binary distributions”, they are compiled into the python byte code for specific architectures and versions of python.

This will also pull in all of the dependencies, and allow for an offline install later.

```
# Example of pulling in the desired public plugins as well
PUB="peek-plugin-noop"
PUB="$PUB peek-core-user"
PUB="$PUB peek-plugin-active-task"
PUB="$PUB peek-plugin-chat"

# Private Plugins
PRI=`ls ../*.tar.gz`

# Build the wheels
pip wheel --no-cache --find-links ../ $PRI $PUB
```

Zip the plugin dist dir.

Windows:

```
cd ~
tar cvjf plugin-win.tar.bz2 -C ~/plugin-release-dir plugin-win
```

Linux:

```
cd ~
tar cvjf plugin-linux.tar.bz2 -C ~/plugin-release-dir plugin-linux
```

Cleanup the release directory:

```
rm -rf cd ~/plugin-release-dir
```

8.3.2 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

8.4 Develop Peek Platform

Warning: This document extends, *Setup OS Requirements Windows* or the *Setup OS Requirements Linux* depending on your OS.

Most development will be for the plugins, not platform, so these instructions are not high priority.

Synerty uses PyCharm as its choice of IDE and Git management tool.

GitLab to manage and share your Git repositories: <https://gitlab.synerty.com>

Note: The reader needs be familiar with, or will become familiar with the following:

- [GIT](#)
- [Python 3.6+](#)
- [Python Twisted](#)
- [HTML](#)
- [SCSS](#)
- [Ant Design](#)
- [TypeScript](#)
- [Angular](#) (Angular2+, not AngularJS aka Angular1)
- [CapacitorJS](#)

Note: This a cross platform development guide, all commands are written for bash.

Bash is installed by default on Linux.

Windows users should use bash from msys, which comes with git for windows, *Setup Msys Git*.

8.4.1 Development Setup Objective

This guide lists the synerty-peek repositories that can be cloned and how to clone. The document contains instructions for obtaining the dependencies, building the front end packages and Building synerty-peek for development or production.

There is assumed understanding of *git*, forking and committing.

8.4.2 Hardware Recommendation

- 32gb of ram (minimum 16gb)

8.4.3 Software Installation and Configuration

On a Windows machine the follow commands will be run using the bash shell, see *Setup Msys Git*.

Synerty Peek Repositories

Synerty's Repositories:

- synerty-peek
- peek-plugin-base
- peek-plugin-base-js
- peek-agent-service
- peek-office-service
- peek-field-app
- peek-platform
- peek-logic-service
- peek-admin-app
- peek-worker-service

Fork Peek Repositories

Create a GitLab group using your hyphenated full name as the namespace, i.e.

<https://gitlab.synerty.com/john-smith>

Create a GitLab subgroup under your namespace named peek, i.e.

<https://gitlab.synergy.com/john-smith/peek>

Create a GitLab subgroup under your namespace named peek-util, i.e.

<https://gitlab.synergy.com/john-smith/peek-util>

Create a GitLab access token with a 24 hour expiry :

https://gitlab.synergy.com/profile/personal_access_tokens

Fork all repositories from <https://gitlab.synergy.com/peek> to your namespace/peek subgroup:

```
export ACCESS_TOKEN="" # https://gitlab.synergy.com/profile/personal_access_tokens
export GROUP_ID="2" # https://gitlab.synergy.com/peek
export YOUR_SUBGROUP_ID="" # Your namespace/peek subgroup id to fork to

function loadProjectIds() {
    curl --location --request GET "https://gitlab.synergy.com/api/v4/groups/${GROUP_ID}/projects?per_page=100" \
        --header 'Content-Type: application/json' \
        --header "Authorization: Bearer ${ACCESS_TOKEN}" \
        --header 'Content-Type: application/json' \
        --data-raw '' | jq '.[].id'
}

for REPO_ID in `loadProjectIds`
do
    curl --location --request POST "https://gitlab.synergy.com/api/v4/projects/${REPO_ID}/fork" \
        --header 'Content-Type: application/json' \
        --header "Authorization: Bearer ${ACCESS_TOKEN}" \
        --data-raw '{"id": "${ID}", "namespace": "${YOUR_SUBGROUP_ID}'}'
done
```

Fork all repositories from <https://gitlab.synergy.com/peek-util> to your namespace/peek-util subgroup:

```
export ACCESS_TOKEN="" # https://gitlab.synergy.com/profile/personal_access_tokens
export GROUP_ID="26" # https://gitlab.synergy.com/peek-util
export YOUR_SUBGROUP_ID="" # Your namespace/peek-util subgroup id to fork to

function loadProjectIds() {
    curl --location --request GET "https://gitlab.synergy.com/api/v4/groups/${GROUP_ID}/projects?per_page=100" \
        --header 'Content-Type: application/json' \
        --header "Authorization: Bearer ${ACCESS_TOKEN}" \
        --header 'Content-Type: application/json' \
        --data-raw '' | jq '.[].id'
}

for REPO_ID in `loadProjectIds`
do
    curl --location --request POST "https://gitlab.synergy.com/api/v4/projects/${REPO_ID}/fork" \
        --header 'Content-Type: application/json' \
        --header "Authorization: Bearer ${ACCESS_TOKEN}" \
        --data-raw '{"id": "${ID}", "namespace": "${YOUR_SUBGROUP_ID}'}'
done
```

Clone all of the projects in your namespace/peek subgroup to ~/dev-peek/:

```

export ACCESS_TOKEN="" # https://gitlab.synerty.com/profile/personal_access_tokens
export YOUR_NAMESPACE="" # Your GitLab namespace group, i.e. "john-smith"
export YOUR_SUBGROUP_ID="" # Your GitLab namespace/peek subgroup id
export DIR=~/.dev-peek"

function loadProjectIds() {
    curl --location --request GET "https://gitlab.synerty.com/api/v4/groups/${YOUR_
↪SUBGROUP_ID}/projects?per_page=100" \
        --header 'Content-Type: application/json' \
        --header "Authorization: Bearer ${ACCESS_TOKEN}" \
        --header 'Content-Type: application/json' \
        --data-raw '' | jq '.[].name'
}

if [ ! -d ${DIR} ]; then
    mkdir ${DIR}
    cd ${DIR}
    for REPO_NAME in `loadProjectIds`
    do
        NAME="${REPO_NAME%\"}"
        NAME="${NAME#\"}"
        URL=https://gitlab.synerty.com/${YOUR_NAMESPACE}/${NAME}.git
        echo $URL
        git clone $URL
    done
fi

```

Clone all of the projects in your namespace/peek-util subgroup to ~/.dev-peek-util/:

```

export ACCESS_TOKEN="" # https://gitlab.synerty.com/profile/personal_access_tokens
export YOUR_NAMESPACE="" # Your GitLab namespace group, i.e. "john-smith"
export YOUR_SUBGROUP_ID="" # Your GitLab namespace/peek subgroup id
export DIR=~/.dev-peek-util"

function loadProjectIds() {
    curl --location --request GET "https://gitlab.synerty.com/api/v4/groups/${YOUR_
↪SUBGROUP_ID}/projects?per_page=100" \
        --header 'Content-Type: application/json' \
        --header "Authorization: Bearer ${ACCESS_TOKEN}" \
        --header 'Content-Type: application/json' \
        --data-raw '' | jq '.[].name'
}

if [ ! -d ${DIR} ]; then
    mkdir ${DIR}
    cd ${DIR}
    for REPO_NAME in `loadProjectIds`
    do
        NAME="${REPO_NAME%\"}"
        NAME="${NAME#\"}"
        URL=https://gitlab.synerty.com/${YOUR_NAMESPACE}/${NAME}.git
        echo $URL
        git clone $URL
    done
fi

```

Note: core.symlink: If false, symbolic links are checked out as small plain files that contain the link text. The default

is true, except *git-clone* or *git-init* will probe and set `core.symlinks` false if appropriate when the repository is created.

Setup Cloned Repositories For Development

Run `setup.py` in all of the repositories located in `~/dev-peek/`:

```
set -e

cd ~/dev-peek
for DIR in */; do
    cd "$DIR"
    NAME=${PWD##*/}
    echo "$NAME"
    pip uninstall -y "$NAME"
    python setup.py develop
    cd ..
done
```

Run `setup.py` in all of the repositories located in `~/dev-peek-util/`:

```
set -e

cd ~/dev-peek-util
for DIR in */; do
    cd "$DIR"
    NAME=${PWD##*/}
    echo "$NAME"
    pip uninstall -y "$NAME"
    python setup.py develop
    cd ..
done
```

Install Front End Modules

Remove the old npm modules files and re-install for field, office and logic service frontend packages. Run the following commands:

```
cd ~/dev-peek/peek-field-app/peek_field_app
[ -d node_modules ] && rm -rf node_modules
npm i
cd ~/dev-peek/peek-office-app/peek_office_app
[ -d node_modules ] && rm -rf node_modules
npm i
cd ~/dev-peek/peek-admin-app/peek_admin_app
[ -d node_modules ] && rm -rf node_modules
npm i
```

Configure Peek Field, Office, Logic Service Settings

Open the config file located at `~/peek/peek-field-service.home/config.json`

Set the property `frontend.docBuildEnabled` to false.

Set the property `frontend.webBuildEnabled` to false.

Open the config file located at `~/peek/peek-office-service.home/config.json`

Set the property `frontend.docBuildEnabled` to `false`.

Set the property `frontend.webBuildEnabled` to `false`.

Open the config file located at `~/peek/peek-logic-service.home/config.json`

Set the property `frontend.docBuildEnabled` to `false`.

Set the property `frontend.webBuildEnabled` to `false`.

Set the property `httpServer.admin.recovery_user.username` to `“recovery”`.

Set the property `httpServer.admin.recovery_user.password` to `“synerty”`.

Compile Front End Packages For Development

Run the following commands in separate terminal sessions:

```
# Terminal 1
cd ~/dev-peek/peek-field-app/peek_field_app
ng build --watch

# Terminal 2
cd ~/dev-peek/peek-admin-app/peek_admin_app
ng build --watch

# Terminal 3
cd ~/dev-peek/peek-office-app/peek_office_app
ng build --watch

# Terminal 4
run_peek_logic_service

# Terminal 5
run_peek_office_service
```

Viewing Peek Services In The Browser

Peek Mobile: <http://localhost:8000>

Peek Desktop: <http://localhost:8002>

Peek Admin: <http://localhost:8010>

8.4.4 What Next?

Refer back to the *[How to Use Peek Documentation](#)* guide to see which document to follow next.

8.5 Publish Peek Platform

8.5.1 Building a Production Release

The peek package has build scripts that generate a platform build

Open the command prompt and enter the bash shell

Change root directory of synerty-peek, example:

```
cd ~peek/dev-peek/synerty-peek/
```

Check and take note of synerty-peek's latest release version on [pypi](#)

Run the follow command being sure to increment the version number:

```
./publish_platform.sh #.#.##
```

Note: Prod build, it tags, commits and test uploads to [testpypi](#).

Run the following script to upload the new release to [pypi](#):

```
./pypi_upload.sh
```

8.5.2 Building a Development Release

The peek package has build scripts that generate a development build

Open the command prompt and enter the bash shell

Change root directory of synerty-peek, example:

```
cd ~peek/dev-peek/synerty-peek/
```

Run the follow command being sure to increment the version number:

```
./publish_platform.sh #.#.#.dev#
```

Note: Dev build, it doesn't tag, commit or test upload, but still generates a build.

Warning: Omitting the dot before dev will cause the script to fail as setuptools adds the dot in if it's not there, which means the cp commands won't match files.

8.5.3 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

8.6 Package Peek Platform

Note: The Windows or Linux requirements must be followed before following this guide.

To install the peek platform, you may use a Synerty provided release or build your own.

A release is a zip file containing all the required node_modules and python packages.

8.6.1 Building a Windows Release

This section contains the steps to build your own platform release.

Ensure that msys git is installed. *Setup Msys Git*.

The python package scripts use git to detect git ignored files.

Open a PowerShell window.

Create and change to a working directory where you're happy for the release to be created.

```
Set-Location C:\Users\peek
```

Download the platform build script. Run the following commands in the power shell window.

```
$file = "package_platform_win.ps1";
$uri = "https://bitbucket.org/synerty/synerty-peek/raw/master/scripts/win/$file";
[Net.ServicePointManager]::SecurityProtocol = "tls12, tls11, tls";
Invoke-WebRequest -Uri $uri -UseBasicParsing -OutFile $file;
```

Note: If you get a big red error that reads:

Invoke-WebRequest : The request was aborted: Could not create SSL/TLS secure channel.

Then download and use the latest version of PowerShell

<https://github.com/PowerShell/PowerShell/releases/download/v6.1.1/PowerShell-6.1.1-win-x64.msi>

Run the platform build script.

```
PowerShell.exe -ExecutionPolicy Bypass -File $file <version>
```

Where <version> is the release you wish to build, for example 1.3.3

The script will download the latest peek platform release and all its dependencies.

Take note of the end of the script, it will print out where the release is.

8.6.2 Building a Linux Release

This section contains the steps to build your own platform release.

Download the platform build script. Run the following commands in the power shell window.

```
file="package_platform_linux.sh";  
uri="https://bitbucket.org/synerty/synerty-peek/raw/master/scripts/linux/$file";  
wget $uri
```

Run the platform build script.

```
bash $file <version>
```

Where <version> is the release you wish to build, for example 1.3.3

The script will download the latest peek platform release and all its dependencies.

Take note of the end of the script, it will print out where the release is.

8.6.3 Building a macOS Release

This section contains the steps to build your own platform release.

Download the platform build script. Run the following commands in the power shell window.

```
file="package_platform_macos.sh";  
uri="https://bitbucket.org/synerty/synerty-peek/raw/master/scripts/macOS/$file";  
curl -O $uri
```

Run the platform build script.

```
bash $file <version>
```

Where <version> is the release you wish to build, for example 1.3.3

The script will download the latest peek platform release and all its dependencies.

Take note of the end of the script, it will print out where the release is.

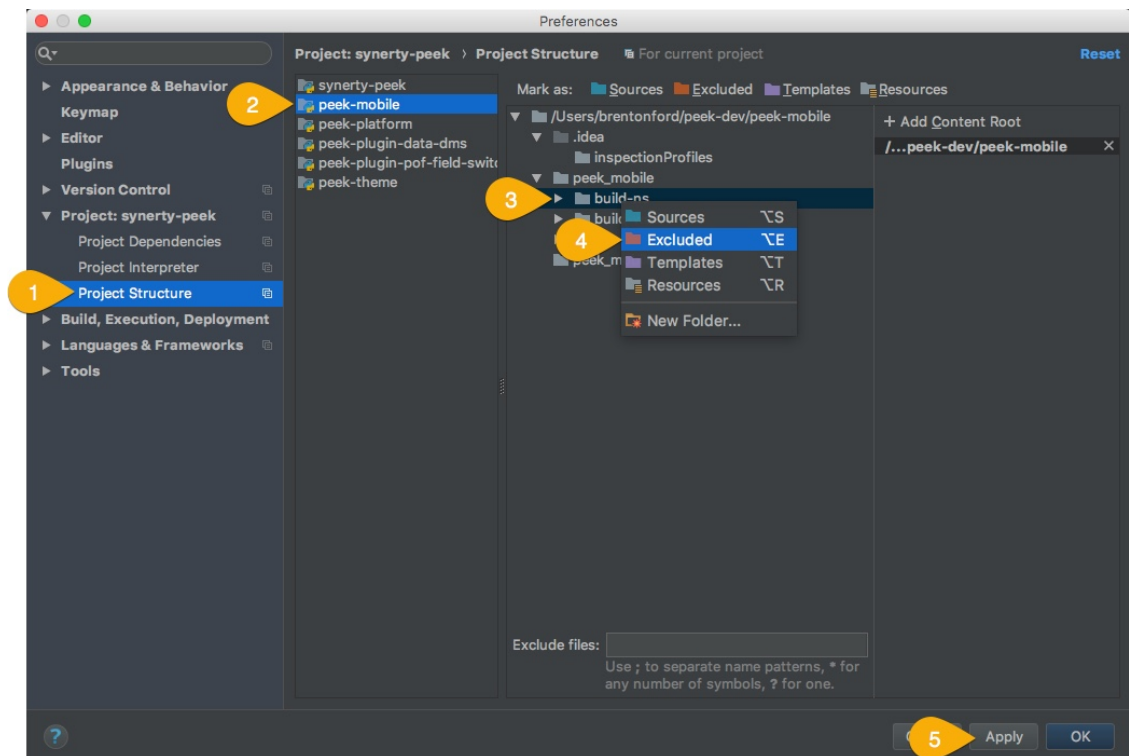
8.6.4 What Next?

Refer back to the [How to Use Peek Documentation](#) guide to see which document to follow next.

8.7 Setup Pycharm IDE

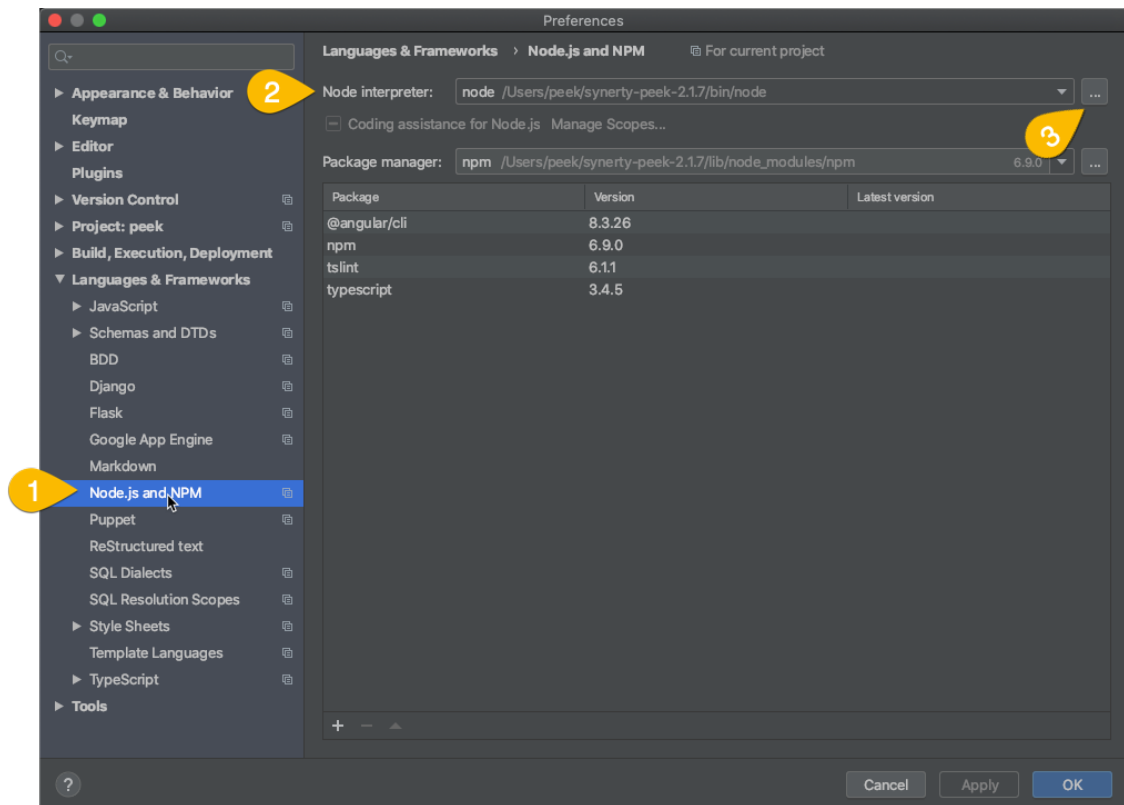
1. Open pycharm,
 1. Open the peek project, open in new window
 2. Open each of the other projects mentioned above, add to current window

2. File -> Settings (Ctrl+Alt+S with eclipse keymap)
 1. Editor -> Inspection (use the search bar for finding the inspections)
 1. Disable Python -> “PEP8 Naming Convention Violation”
 2. Change Python -> “Type Checker” from warning to error
 3. Change Python -> “Incorrect Docstring” from warning to error
 4. Change Python -> “Missing type hinting ...” from warning to error
 5. Change Python -> “Incorrect call arguments” from warning to error
 6. Change Python -> “Unresolved references” from warning to error
 2. Project -> Project Dependencies
 1. peek_platform depends on -> plugin_base
 2. peek_logic_service depends on -> peek_platform, peek_admin_app
 3. peek_office_service depends on -> peek_platform, peek_field_app
 4. peek_agent_service depends on -> peek_platform
 5. peek_worker_service depends on -> peek_platform
 3. Project -> Project Structure
 1. peek-field-app -> Excluded
 2. peek-office-app -> Excluded
 3. peek-admin-app -> Excluded



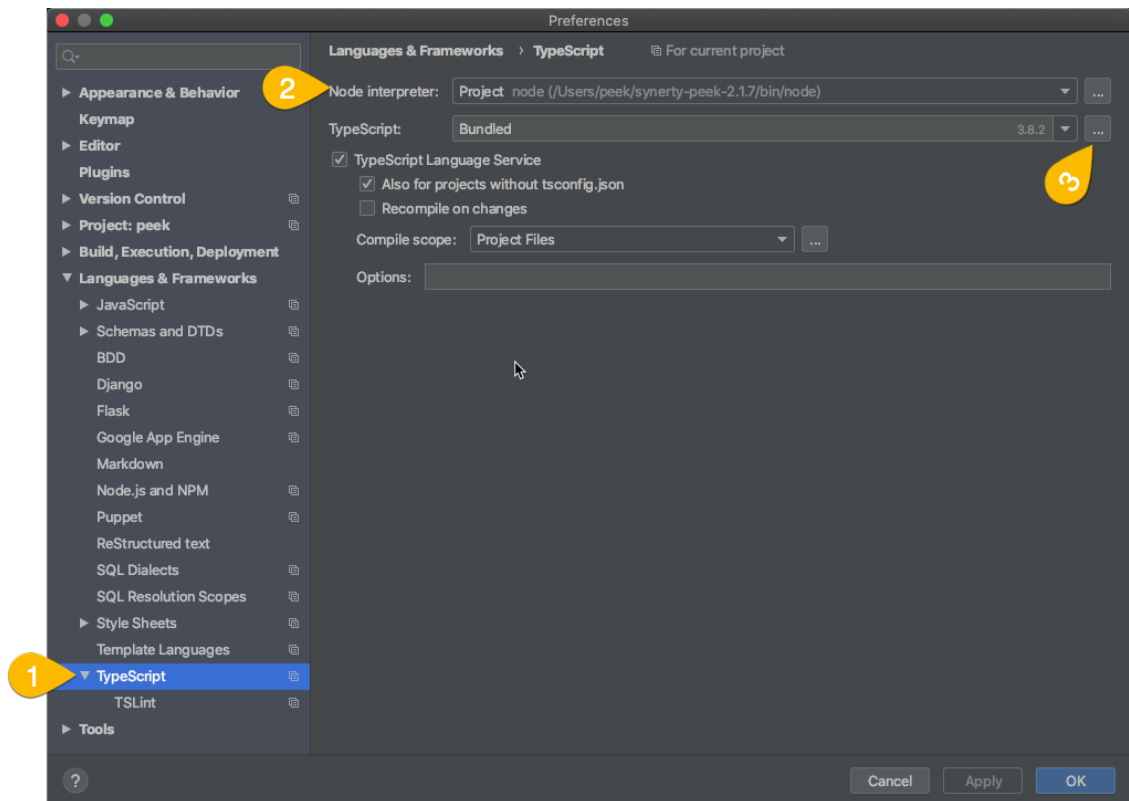
4. Languages & Frameworks -> Node.js and NPM
 1. Node interpreter -> ~/node-v10.20.0/bin/node

2. Remove other node interpreters



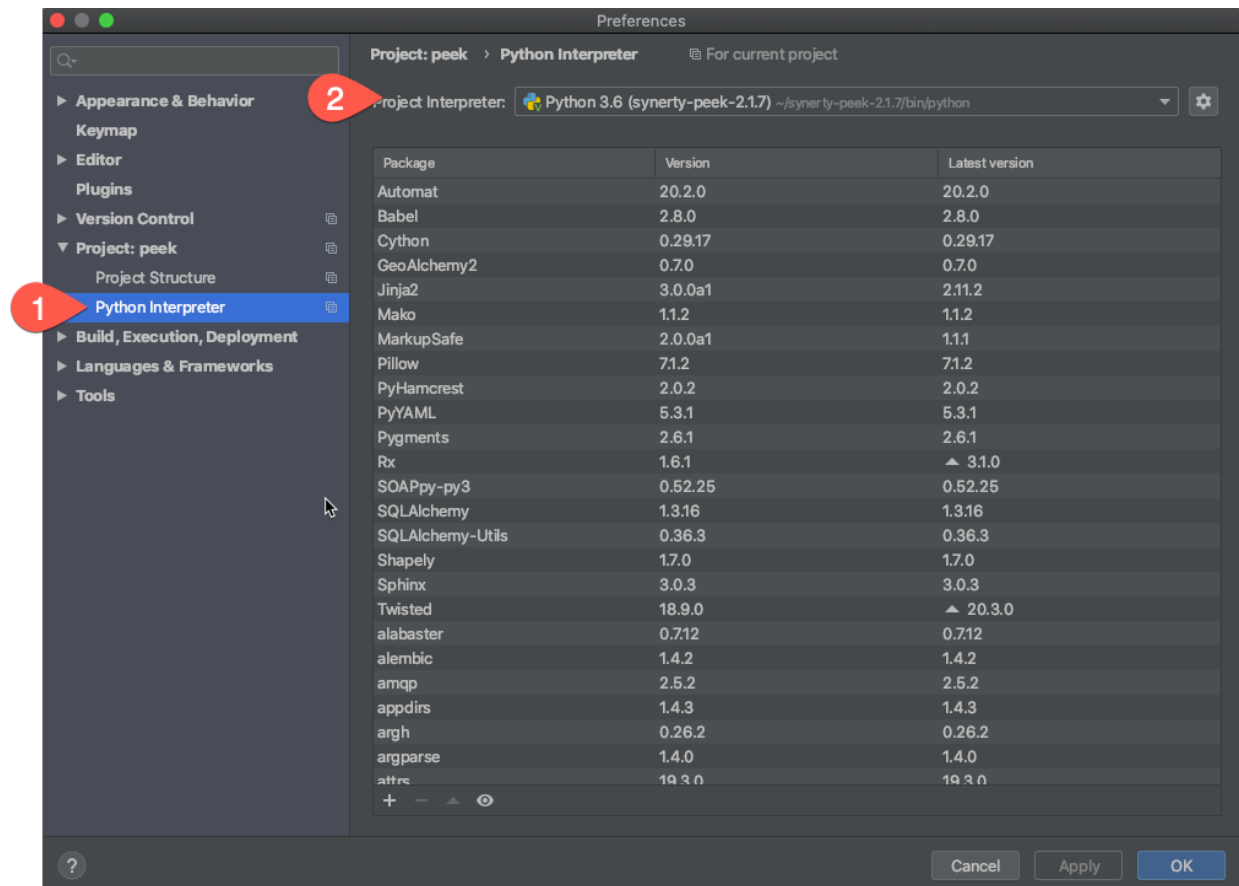
5. Languages & Frameworks -> TypeScript

1. Node interpreter -> ~/node-v14.15.3/bin/node
2. Enable TypeScript Compiler -> Checked
3. Set options manually -> Checked
4. Command line options -> -target es5 --experimentalDecorators --lib es6,dom --sourcemap --emitDecoratorMetadata
5. Generate source maps -> Checked



Configure your developing software to use the virtual environment you wish to use

Here is an example of the setting in PyCharm:



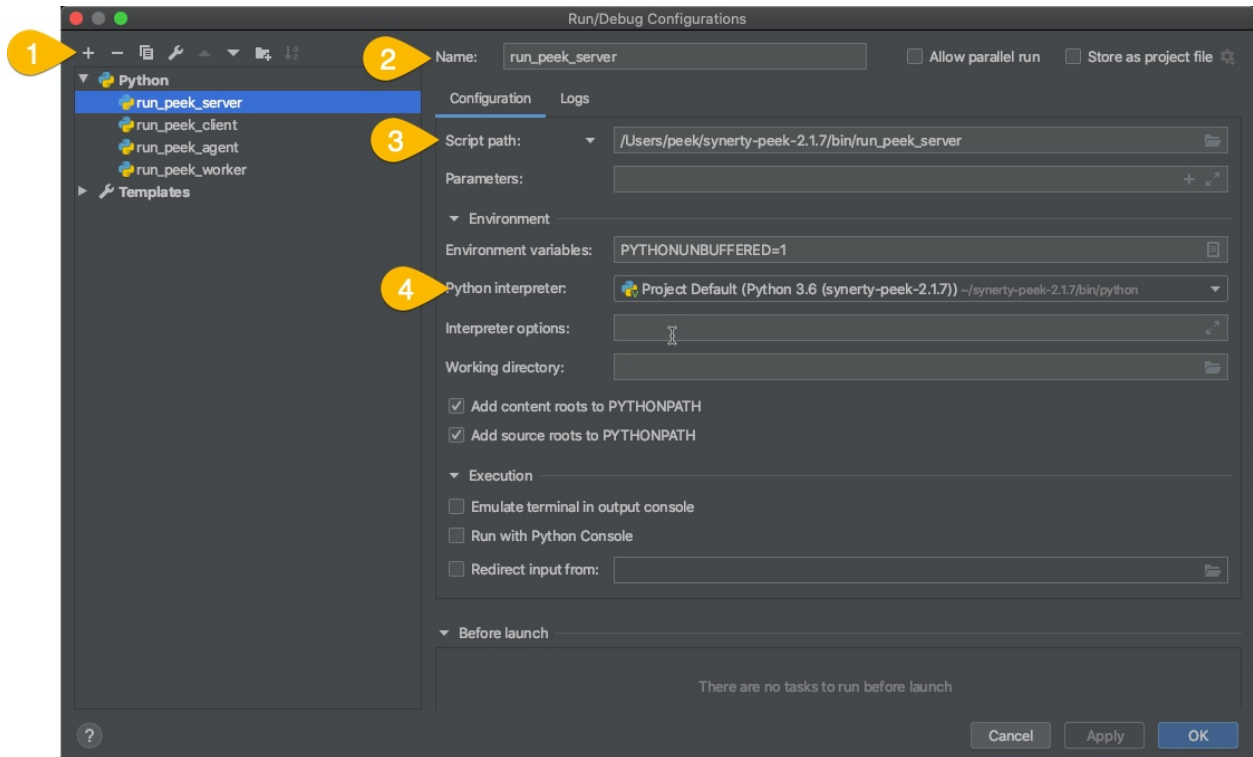
Restart the services that use the plugin

Note: The plugins that aren't being developed should be installed with pip.

This is an example of running the peek logic service in debug mode using **PyCharm**

Under the drop down “Run” then “Edit Configurations...”

1. Add new configuration, select “Python”
2. Update the “Name:”
3. Locate the script you wish to run
4. Check that the “Python Interpreter” is correct



8.8 Setup VS Code IDE

1. Visual Studio Code,

Download <https://code.visualstudio.com>

Add PATH to environment variables

```
"C:\Program Files (x86)\Microsoft VS Code\bin"
```

8.9 Helper Scripts for Development

There is a `dev-scripts` folder under `synerty-peek` project, where you can find helper scripts to assist your development of Peek.

8.9.1 `git_check_sync.py`

This script checks synchronisation status between the official Peek repositories and your forked repositories, your forked repositories and your local git `dev-peek` codebase.

Usage:

```
python git_check_sync.py
```

Please create an access token on gitlab if you haven't owned one. Please go to https://gitlab.synerty.com/profile/personal_access_tokens. Type a name and select an expiration date. Please make sure you check the following scopes: `api`, `read_user`, `read_repository`. This means your token only has read access to gitlab.

You copy your token string to `.gitlabtoken` file in the root of `synerty-peek` project.

Run the script by `python git_check_sync.py`.

The scripts will start off by checking the difference between your forked projects on gitlab and the official repositories. Then, it will check your forked ones with your local git folders. Any difference in these 2 checks will be prompted in terminal and offer you a choice to proceed or abort the folder-by-folder synchronisation check happening next.

If you type anything other than `y`, it abort the checks. Otherwise, it will proceed to folder-by-folder git synchronisation check which takes a couple of minutes. In this check, it validates that the remote master branches in your forks on gitlab are in sync of their counterparts in official repository on gitlab (ORIGIN IN SYNC); that the master branches in your local git folders are in sync of their remote branches in respective forks (LOCAL IN SYNC).

A summary will display by category of check types with repository info in each. Example:

```
===== Summary =====
{'localNotInSync': [{1888: 'https://gitlab.synerty.com/your-name/peek/enterprise/peek-
↪plugin-enmac-field-incidents'},
                    {1887: 'https://gitlab.synerty.com/your-name/peek/enterprise/peek-
↪plugin-enmac-field-assessments'},
                    {1860: 'https://gitlab.synerty.com/your-name/peek/community/peek-
↪core-docdb'},
                    {1856: 'https://gitlab.synerty.com/your-name/peek/community/peek-
↪plugin-diagram'},
                    {1850: 'https://gitlab.synerty.com/your-name/peek/community/peek-
↪field-app'}]},
'originNotInSync': []}
```

8.9.2 git_check_branch.sh

This script shows your local branch status. It is like a `git status` for all repositories in `dev-peek`. It displays all repositories that are not on `master` branch with the number of commits unpushed, the number of files untracked to git, and the number of changes made but not committed.

Usage:

```
bash git_check_branch.sh
```

8.9.3 git_optimize.sh

This script collects garbage and optimises performace in all your `dev-peek` projects.

Usage:

```
bash git_optimize.sh
```


8.10 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

9.1 Troubleshooting Windows

9.1.1 Test cx_Oracle in Python

Use the following instructions to test the installaion of cx_Oracle

Open the “Python 3.5 (64-bit)” application from the windows start menu.

Run the following commands in Python:

```
import cx_Oracle
con = cx_Oracle.connect('username/password@hostname/instance')
print con.version
# Exppect to see "12.1.0.2.0"
con.close()
```

9.2 Troubleshooting Debian

9.2.1 Test cx_Oracle in Python

Use the following instructions to test the installaion of cx_Oracle

Login as peek and run:

```
python
```

Run the following commands in Python:

```
import cx_Oracle
con = cx_Oracle.connect('username/password@hostname/instance')
print con.version
# Expect to see "12.1.0.2.0"
con.close()
```

9.2.2 OSError: inotify instance limit reached

This is caused when developing peek. The Peek Platform watches the files in each plugin and then copies them the the UI build directories as they change, EG build-web.

There are quite a few files to monitor and the limits are nice and conservative on Linux by default

To solve this problem, run the following command as root

```
echo "fs.inotify.max_user_instances=2048" >> /etc/sysctl.conf
echo "fs.inotify.max_user_watches=524288" >> /etc/sysctl.conf
sysctl -p
```

9.3 Troubleshooting macOS

9.3.1 Test cx_Oracle in Python

Use the following instructions to test the installaion of cx_Oracle

Open the “Python 3.5 (64-bit)” application from the windows start menu.

Run the following commands in Python:

```
import cx_Oracle
con = cx_Oracle.connect('username/password@hostname/instance')
print con.version
# Expect to see "12.1.0.2.0"
con.close()
```

9.3.2 ORA-21561: OID generation failed

In macOS, You might see the following error :

```
sqlalchemy.exc.DatabaseError: (cx_Oracle.DatabaseError) ORA-21561: OID generation_
↪ failed
```

This is caused by the macOS hostname under “sharing” not matching the name in /etc/hosts

Run hostname to get the name of the mac :

```
Synerty-256:build-web jchesney$ hostname  
syn256.local
```

Confirm that it matches the hostnames for 127.0.0.1 and ::1 in /etc/hosts :

```
Synerty-256:build-web jchesney$ cat /etc/hosts  
##  
# Host Database  
#  
# localhost is used to configure the loopback interface  
# when the system is booting. Do not change this entry.  
##  
127.0.0.1        localhost syn256.local  
255.255.255.255 broadcasthost  
::1             localhost syn256.local
```


CHAPTER 10

Utilities

11.1 File `plugin_package.json`

This page will describe the options in the `plugin_package.json`

11.2 SynertyPeek

11.2.1 `peek_plugin_base`

(P) agent

(M) `PeekAgentPlatformHookABC`

```
class peek_plugin_base.agent.PeekAgentPlatformHookABC.PeekAgentPlatformHookABC
    Bases: peek_plugin_base.PeekPlatformCommonHookABC.PeekPlatformCommonHookABC,
           peek_plugin_base.PeekPlatformServerInfoHookABC.PeekPlatformServerInfoHookABC,
           peek_plugin_base.agent.PeekPlatformAgentHttpHookABC.PeekPlatformAgentHttpHookABC
```

(M) `PeekPlatformAgentHttpHookABC`

```
class peek_plugin_base.agent.PeekPlatformAgentHttpHookABC.PeekPlatformAgentHttpHookABC
    Bases: object
```

Peek Agent Service HTTP External API Hook

The methods provided by this class apply to the HTTP service that provides resources (vortex, etc) between the server and the agent, worker and client.

These resources will not be available to the web apps.

addAgentExternalApiResource (*pluginSubPath: bytes, resource: tx-
httputil.site.BasicResource.BasicResource*) → None

Add Agent Resource

Add a cusotom implementation of a served http resource.

Parameters

- **pluginSubPath** – The resource path where you want to serve this resource.
- **resource** – The resource to serve.

Returns None

rootAgentResource

Agent Root Resource

This returns the root site resource for this plugin.

(M) PluginAgentEntryHookABC

```
class peek_plugin_base.agent.PluginAgentEntryHookABC.PluginAgentEntryHookABC (pluginName: str,  
plug-  
in-  
Root-  
Dir:  
str,  
plat-  
form:  
peek_plugin_base.a)

Bases: peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC

platform

publishedAgentApi
```

(P) client

(M) PeekClientPlatformHookABC

```
class peek_plugin_base.client.PeekClientPlatformHookABC.PeekClientPlatformHookABC

Bases: peek_plugin_base.PeekPlatformCommonHookABC.PeekPlatformCommonHookABC,  
peek_plugin_base.client.PeekPlatformFieldHttpHookABC.  
PeekPlatformFieldHttpHookABC, peek_plugin_base.client.  
PeekPlatformOfficeHttpHookABC.PeekPlatformOfficeHttpHookABC,  
peek_plugin_base.PeekPlatformServerInfoHookABC.PeekPlatformServerInfoHookABC,  
peek_plugin_base.PeekPlatformFileStorageHookABC.PeekPlatformFileStorageHookABC
```

(M) PeekPlatformFieldHttpHookABC

```
class peek_plugin_base.client.PeekPlatformFieldHttpHookABC.PeekPlatformFieldHttpHookABC

Bases: object

Peek Platform Site HTTP Hook
```

The methods provided by this class apply to the HTTP sites served by the Client service for the mobile and desktop apps, and the Server service for the admin app.

It is not the HTTP service that provides resources (vortex, etc) between the server and the agent, worker and client.

addFieldResource (*pluginSubPath: bytes, resource: txhttputil.site.BasicResource.BasicResource*)

→ None

Add Site Resource

Add a cusotom implementation of a served http resource.

Parameters

- **pluginSubPath** – The resource path where you want to serve this resource.
- **resource** – The resource to serve.

Returns None

addFieldStaticResourceDir (*dir: str*) → None

Add Site Static Resource Directory

Calling this method sets up directory `dir` to be served by the site.

Parameters **dir** – The file system directory to be served.

Returns None

rootFieldResource

Site Root Resource

This returns the root site resource for this plugin.

(M) PeekPlatformOfficeHttpHookABC

class `peek_plugin_base.client.PeekPlatformOfficeHttpHookABC.PeekPlatformOfficeHttpHookABC`

Bases: object

Peek Platform Site HTTP Hook

The methods provided by this class apply to the HTTP sites served by the Client service for the mobile and desktop apps, and the Server service for the admin app.

It is not the HTTP service that provides resources (vortex, etc) between the server and the agent, worker and client.

addOfficeResource (*pluginSubPath: bytes, resource: txhttputil.site.BasicResource.BasicResource*)

→ None

Add Site Resource

Add a cusotom implementation of a served http resource.

Parameters

- **pluginSubPath** – The resource path where you want to serve this resource.
- **resource** – The resource to serve.

Returns None

addOfficeStaticResourceDir (*dir: str*) → None

Add Site Static Resource Directory

Calling this method sets up directory `dir` to be served by the site.

Parameters `dir` – The file system directory to be served.

Returns None

rootOfficeResource

Site Root Resource

This returns the root site resource for this plugin.

(M) PluginClientEntryHookABC

```
class peek_plugin_base.client.PluginClientEntryHookABC.PluginClientEntryHookABC(pluginName:
                                                                    str,
                                                                    plug-
                                                                    in-
                                                                    Root-
                                                                    Dir:
                                                                    str,
                                                                    plat-
                                                                    form:
                                                                    peek_plugin_base
```

Bases: `peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC`

angularFrontendAppDir

Angular Frontend Dir

This directory will be linked into the angular app when it is compiled.

Returns The absolute path of the Angular2 app directory.

angularMainModule

Angular Main Module

Returns The name of the main module that the Angular2 router will lazy load.

platform

publishedClientApi

(P) server

(M) PeekPlatformAdminHttpHookABC

```
class peek_plugin_base.server.PeekPlatformAdminHttpHookABC.PeekPlatformAdminHttpHookABC
Bases: object
```

Peek Platform Site HTTP Hook

The methods provided by this class apply to the HTTP sites served by the Client service for the mobile and desktop apps, and the Server service for the admin app.

It is not the HTTP service that provides resources (vortex, etc) between the server and the agent, worker and client.

addAdminResource (*pluginSubPath: bytes, resource: txhttputil.site.BasicResource.BasicResource*)

→ None

Add Site Resource

Add a cusotom implementation of a served http resource.

Parameters

- **pluginSubPath** – The resource path where you want to serve this resource.
- **resource** – The resource to serve.

Returns None

addAdminStaticResourceDir (*dir: str*) → None

Add Site Static Resource Directory

Calling this method sets up directory `dir` to be served by the site.

Parameters **dir** – The file system directory to be served.

Returns None

rootAdminResource

Site Root Resource

This returns the root site resource for this plugin.

(M) PeekPlatformServerHttpHookABC

class `peek_plugin_base.server.PeekPlatformServerHttpHookABC.PeekPlatformServerHttpHookABC`

Bases: `object`

Peek Platform Server HTTP Hook

The methods provided by this class apply to the HTTP service that provides resources (vortex, etc) between the server and the agent, worker and client.

These resources will not be available to the web apps.

addServerResource (*pluginSubPath: bytes, resource: txhttputil.site.BasicResource.BasicResource*) → None

Add Server Resource

Add a cusotom implementation of a served http resource.

Parameters

- **pluginSubPath** – The resource path where you want to serve this resource.
- **resource** – The resource to serve.

Returns None

addServerStaticResourceDir (*dir: str*) → None

Add Server Static Resource Directory

Calling this method sets up directory `dir` to be served by the site.

Parameters **dir** – The file system directory to be served.

Returns None

rootServerResource

Server Root Resource

This returns the root site resource for this plugin.

(M) PeekServerPlatformHookABC

```
class peek_plugin_base.server.PeekServerPlatformHookABC.PeekServerPlatformHookABC
    Bases: peek_plugin_base.PeekPlatformCommonHookABC.PeekPlatformCommonHookABC,
           peek_plugin_base.server.PeekPlatformAdminHttpHookABC.
           PeekPlatformAdminHttpHookABC,                               peek_plugin_base.server.
           PeekPlatformServerHttpHookABC.PeekPlatformServerHttpHookABC,
           peek_plugin_base.PeekPlatformFileStorageHookABC.PeekPlatformFileStorageHookABC
```

dbConnectionString

DB Connect String

Returns The SQLAlchemy database engine connection string/url.

(M) PluginLogicEntryHookABC

```
class peek_plugin_base.server.PluginLogicEntryHookABC.PluginLogicEntryHookABC (pluginName:
                                                                 str,
                                                                 plug-
                                                                 in-
                                                                 Root-
                                                                 Dir:
                                                                 str,
                                                                 plat-
                                                                 form:
                                                                 peek_plugin_base.
```

Bases: *peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC*

dbSession

Database Session

Returns An instance of the sqlalchemy ORM session

migrateStorageSchema (*metadata: sqlalchemy.sql.schema.MetaData*) → None

Initialise the DB

Parameters metadata – the SQLAlchemy metadata for this plugins schema

platform

publishedServerApi

Published Server API

:return class that implements the API that can be used by other PLUGINS on this platform.

(M) PluginServerStorageEntryHookABC

```
class peek_plugin_base.server.PluginServerStorageEntryHookABC.PluginServerStorageEntryHookABC
    Bases: object
```

dbEngine

DB Engine

This is a helper property that can be used by the papp to get easy access to the SQLAlchemy C{Engine}

Returns The instance of the database engine for this plugin

dbMetadata

DB Metadata

This property returns an instance to the metadata from the ORM Declarative on which, all the ORM classes have inherited.

This means the metadata knows about all the tables.

NOTE: The plugin must be constructed with a schema matching the plugin package

Returns The instance of the metadata for this plugin.

Example from peek_plugin_noop.storage.DeclarativeBase.py

```
metadata = MetaData(schema="noop")
DeclarativeBase = declarative_base(metadata=metadata)
```

dbSessionCreator

Database Session

This is a helper property that can be used by the papp to get easy access to the SQLAlchemy C{Session}

Returns An instance of the sqlalchemy ORM session

prefetchDeclarativeIds (*Declarative, count*) → twisted.internet.defer.Deferred

Get PG Sequence Generator

A PostgreSQL sequence generator returns a chunk of IDs for the given declarative.

Returns A generator that will provide the IDs

Return type an iterator, yielding the numbers to assign

(M) PluginServerWorkerEntryHookABC

```
class peek_plugin_base.server.PluginServerWorkerEntryHookABC.PluginServerWorkerEntryHookABC
Bases: object
```

(P) storage

(M) AlembicEnvBase

```
class peek_plugin_base.storage.AlembicEnvBase.AlembicEnvBase(targetMetadata)
Bases: object
```

run()

Run migrations in ‘online’ mode. In this scenario we need to create an Engine and associate a connection with the context.

```
peek_plugin_base.storage.AlembicEnvBase.ensureSchemaExists(engine, schemaName)
```

```
peek_plugin_base.storage.AlembicEnvBase.isMssqlDialect(engine)
```

```
peek_plugin_base.storage.AlembicEnvBase.isPostgreSQLDialect(engine)
```

(M) DbConnection

```
class peek_plugin_base.storage.DbConnection.DbConnection (dbConnectString:
                                                                str,          metadata:
                                                                sqlalchemy.sql.schema.MetaData,
                                                                alembicDir:      str,
                                                                dbEngineArgs:    Optional[Dict[str,
                                                                Union[str, int]]] =
                                                                None,            enableFor-
                                                                eignKeys=False,  en-
                                                                ableCreateAll=True)
```

Bases: object

SQLAlchemy Database Connection

This class takes care of migrating the database and establishing thing database connections and ORM sessions.

Parameters

- **dbConnectString** – The connection string for the DB. See <http://docs.sqlalchemy.org/en/latest/core/engines.html>
- **metadata** – The instance of the metadata for this connection, This is schema qualified `MetaData(schema="schama_name")`
- **alembicDir** – The absolute location of the alembic directory (versions dir lives under this)
- **dbEngineArgs** – The arguments to pass to the database engine, See <http://docs.sqlalchemy.org/en/latest/core/engines.html#engine-creation-api>
- **enableCreateAll** – If the schema doesn't exist, then the migration is allowed to use `matadata.create_all()`
- **enableForeignKeys** – Perform a check to ensure foriegn keys have indexes after the db is migrated and connected.

checkForeignKeys (*engine: sqlalchemy.engine.base.Engine*) → None

Check Foreign Keys

Log any foreign keys that don't have indexes assigned to them. This is a performance issue.

closeAllSessions ()

Close All Session

Close all ORM sessions connected to this DB engine.

dbEngine

Get DB Engine

This is not thread safe, use the `ormSession` to execute SQL statements instead. `self.ormSession.execute(...)`

Returns the DB Engine used to connect to the database.

migrate () → None

Migrate

Perform a database migration, upgrading to the latest schema level.

ormSessionCreator

Get Orm Session

Returns A SQLAlchemy session scoped for the callers thread..

prefetchDeclarativeIds (***kwargs*)

`peek_plugin_base.storage.DbConnection.convertToCoreSqlaInsert` (*ormObj, Declarative*)

`peek_plugin_base.storage.DbConnection.pgCopyInsert` (*rawConn, table, inserts*)

(M) LoadPayloadPgUtil

class `peek_plugin_base.storage.LoadPayloadPgUtil.LoadPayloadTupleResult` (*count, encodedPayload*)

Bases: `tuple`

Create new instance of `LoadPayloadTupleResult(count, encodedPayload)`

count

Alias for field number 0

encodedPayload

Alias for field number 1

`peek_plugin_base.storage.LoadPayloadPgUtil.getTuplesPayload` (*logger: logging.Logger, dbSessionCreator: Callable[[sqlalchemy.orm.session.Session], sqlalchemy.sql.selectable.Select, sqlCoreLoadTupleClassmethod: Callable, payloadFilter: Optional[Dict[KT, VT]] = None, fetchSize=50) → twisted.internet.defer.Deferred*)

```
peek_plugin_base.storage.LoadPayloadPgUtil.getTuplesPayloadBlocking (dbSessionCreator:
    Callable[[],
    sqlalchemy.orm.session.Session],
    sql:
    sqlalchemy.sql.selectable.Select,
    sql-
    CoreLoad-
    Tuple-
    Class-
    method:
    Callable,
    payload-
    Filt: Op-
    tional[Dict[KT,
    VT]] =
    None,
    fetch-
    Size=50)
    →
    peek_plugin_base.storage.LoadPa
```

(M) RunPyInPg

```
peek_plugin_base.storage.RunPyInPg.runPyInPg (logger: logging.Logger, db-
    SessionCreator: Callable[[],
    sqlalchemy.orm.session.Session], class-
    MethodToRun: Callable, classMethod-
    ToImportTuples: Optional[Callable],
    *args, **kwargs) → Any

peek_plugin_base.storage.RunPyInPg.runPyInPgBlocking (dbSessionCreator: Callable[[],
    sqlalchemy.orm.session.Session],
    classMethodToRun: Any,
    classMethodToImportTuples:
    Optional[Callable], *args,
    **kwargs) → Any
```

(M) StorageUtil

```
peek_plugin_base.storage.StorageUtil.makeCoreValuesSubqueryCondition (engine,
    col-
    umn,
    values:
    List[Union[int,
    str]])
```

Make Core Values Subquery

Parameters

- **engine** – The database engine, used to determine the dialect
- **column** – The column, eg TableItem.__table__.c.colName
- **values** – A list of string or int values

```
peek_plugin_base.storage.StorageUtil.makeOrmValuesSubqueryCondition (ormSession,
                                                                    column,
                                                                    values:
                                                                    List[Union[int,
                                                                    str]])
```

Make Orm Values Subquery

Parameters

- **ormSession** – The orm session instance
- **column** – The column from the Declarative table, eg `TableItem.colName`
- **values** – A list of string or int values

(M) TypeDecorators

```
class peek_plugin_base.storage.TypeDecorators.PeekLargeBinary (*args, **kwargs)
    Bases: sqlalchemy.sql.type_api.TypeDecorator
```

Peek Large Binary

Construct a `TypeDecorator`.

Arguments sent here are passed to the constructor of the class assigned to the `impl` class level attribute, assuming the `impl` is a callable, and the resulting object is assigned to the `self.impl` instance attribute (thus overriding the class attribute of the same name).

If the class level `impl` is not a callable (the unusual case), it will be assigned to the same instance attribute 'as-is', ignoring those arguments passed to the constructor.

Subclasses can override this to customize the generation of `self.impl` entirely.

bind_expression (*bindvalue*)

Given a bind value (i.e. a `BindParameter` instance), return a SQL expression in its place.

This is typically a SQL function that wraps the existing bound parameter within the statement. It is used for special data types that require literals being wrapped in some special database function in order to coerce an application-level value into a database-specific format. It is the SQL analogue of the `TypeEngine.bind_processor()` method.

The method is evaluated at statement compile time, as opposed to statement construction time.

Note that this method, when implemented, should always return the exact same structure, without any conditional logic, as it may be used in an `executemany()` call against an arbitrary number of bound parameter sets.

See also:

`types_sql_value_processing`

impl

alias of `sqlalchemy.sql.sqltypes.LargeBinary`

(P) worker

(M) CeleryApp

(M) CeleryDbConn

```
peek_plugin_base.worker.CeleryDbConn.getDbEngine()
peek_plugin_base.worker.CeleryDbConn.getDbSession()
peek_plugin_base.worker.CeleryDbConn.prefetchDeclarativeIds(Declarative,
                                                             count) → Optional[Iterable[int]]
```

Prefetch Declarative IDs

This function prefetches a chunk of IDs from a database sequence. Doing this allows us to preallocate the IDs before an insert, which significantly speeds up :

- Orm inserts, especially those using inheritance
- When we need the ID to assign it to a related object that we’re also inserting.

Parameters

- **Declarative** – The SQLAlchemy declarative class. (The class that inherits from DeclarativeBase)
- **count** – The number of IDs to prefetch

Returns An iterable that dispenses the new IDs

```
peek_plugin_base.worker.CeleryDbConn.setConnStringForWindows()
Set Conn String for Windiws
```

Windows has a different way of forking processes, which causes the @worker_process_init.connect signal not to work in “CeleryDbConnInit”

(M) CeleryDbConnInit

```
peek_plugin_base.worker.CeleryDbConnInit.initWorkerConnString(sender,
                                                                **kwargs)
peek_plugin_base.worker.CeleryDbConnInit.initWorkerProcessDbConn(**kwargs)
peek_plugin_base.worker.CeleryDbConnInit.shutdownWorkerProcessDbConn(**kwargs)
peek_plugin_base.worker.CeleryDbConnInit.taskEndCloseSession(**kwargs)
```

(M) PeekWorkerPlatformHookABC

```
class peek_plugin_base.worker.PeekWorkerPlatformHookABC.PeekWorkerPlatformHookABC
Bases: peek_plugin_base.PeekPlatformCommonHookABC.PeekPlatformCommonHookABC
```

(M) PluginWorkerEntryHookABC

class peek_plugin_base.worker.PluginWorkerEntryHookABC.**PluginWorkerEntryHookABC** (*pluginName: str, plugin-Root-Dir: str, platform: peek_plugin_base*)

Bases: *peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC*

celeryAppIncludes
Celery App Includes

This property returns the absolout package paths to the modules with the tasks :Example: ["plugin_noop.worker.NoopWorkerTask"]

Returns A list of package+module names that Celery should import.

platform

(M) PeekPlatformCommonHookABC

class peek_plugin_base.PeekPlatformCommonHookABC.**PeekPlatformCommonHookABC**
Bases: object

getOtherPluginApi (*pluginName: str*) → Optional[object]
Get Other Plugin Api

Asks the plugin for it's api object and return it to this plugin. The API returned matches the platform service.

Parameters **pluginName** – The name of the plugin to retrieve the API for

Returns An instance of the other plugins API for this Peek Platform Service.

serviceId
Service ID

Return a unique identifier for this service.

(M) PeekPlatformFileStorageHookABC

class peek_plugin_base.PeekPlatformFileStorageHookABC.**PeekPlatformFileStorageHookABC**
Bases: object

Peek Platform File Storage Hook

This ABC provides methods allowing plugins to use the file system.

Though there is nothing in place to prevent the plugins doing what ever they like, they should play nice and get their allocated path from here.

fileStorageDirectory
File Storage Directory

This method returns a Path object providing access to the managed file storage location where the plugin can persistently store any files it wants to.

See <https://docs.python.org/3/library/pathlib.html#basic-use>

Returns The plugins managed storage Path object.

(M) PeekPlatformServerInfoHookABC

```
class peek_plugin_base.PeekPlatformServerInfoHookABC.PeekPlatformServerInfoHookABC
    Bases: object
```

Peek Platform Server Info Hook

This ABC provides information for plugins that want to connect to their own code running on the server service, via the inter peek service HTTP.

```
peekServerHost
```

Peek Server Host

Returns The IP address of the server where the peek server service is running.

```
peekServerHttpPort
```

Peek Server HTTP Port

Returns The TCP Port of the Peek Servers HTTP Service (not the admin webapp site)

(M) PeekVortexUtil

```
peek_plugin_base.PeekVortexUtil.peekAdminName = 'peek-admin-app'
```

The vortex name for the Admin browser client

```
peek_plugin_base.PeekVortexUtil.peekAgentName = 'peek-agent-service'
```

The vortex name for the Agent service

```
peek_plugin_base.PeekVortexUtil.peekDesktopName = 'peek-office-app'
```

The vortex name for the Desktop browser clients

```
peek_plugin_base.PeekVortexUtil.peekFieldName = 'peek-field-service'
```

The vortex name for the Field service

```
peek_plugin_base.PeekVortexUtil.peekMobileName = 'peek-field-app'
```

The vortex name for the Mobile device/browser clients

```
peek_plugin_base.PeekVortexUtil.peekServerName = 'peek-logic-service'
```

The vortex name for the Server service

```
peek_plugin_base.PeekVortexUtil.peekStorageName = 'peek-storage-service'
```

The vortex name for the Storage service

```
peek_plugin_base.PeekVortexUtil.peekWorkerName = 'peek-worker-service'
```

The vortex name for the Worker service

(M) PluginCommonEntryHookABC

```
class peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC (pluginName:
                                                                    str,
                                                                    plug-
                                                                    in-
                                                                    Root-
                                                                    Dir:
                                                                    str)
```

Bases: object

load() → None
Load

This will be called when the plugin is loaded, just after the db is migrated. Place any custom initialiaction steps here.

name
Plugin Name

Returns The name of this plugin

packageCfg
Package Config

Returns A reference to the plugin_package.json loader object (see json-cfg)

rootDir
Plugin Root Dir

Returns The absolute directory where the Plugin package is located.

start() → None
Start

This method is called by the platform when the plugin should start

stop() → None
Stop

This method is called by the platform to tell the peek app to shutdown and stop everything it's doing

title
Peek App Title :return the title of this plugin

unload() → None
Unload

This method is called after stop is called, to unload any last resources before the PLUGIN is unlinked from the platform

(M) PluginPackageFileConfig

```
class peek_plugin_base.PluginPackageFileConfig.PluginPackageFileConfig (pluginRootDir:
                                                                    str)
```

Bases: object

This class helps with accessing the config for the plugin_package.json

Constructor

Parameters pluginRootDir – The root directory of this package, where plugin_package.json lives.

config

Config

Returns The jsoncfg config object, for accessing and saving the config.

12.1 PEEP Overviews

12.1.1 PEEP 0-10

PEEP4 Overview

Motivation

The names of Peek plugins were often confusing, as the old “peek_mobile” repository was intended for use in the field, but it can run on either a mobile device or a truck mounted laptop. Therefore, we opted to call this peek_field. peek_field is a WebApp, to make this clearer, the new name is peek_field_app.

PEEP4: Rename Peek Code for v3, Field, Office, App, etc

Significant renaming took place internally as a result of PEEP4. The old and new naming conventions are listed in the following table:

Table 1: Web/Native

v2 Name	V3 Name	Reason
peek-desktop	peek-office-app	This app is intended for use by office staff, this app is desktop and mobile friendly.
peek-mobile	peek-field-app	This app is intended for use by field staff, this app is desktop and mobile friendly.
peek-admin	peek-admin-app	Adding “app” clearly identifies this component as an App.

Table 2: Documentation Projects

v2 Name	V3 Name	Reason
peek-doc-user	peek-field-doc	Renaming this makes it sit next to the web app. People looking at the app will see the doc project as well.
peek-	peek-office-doc	Field and Office could not share a documentation project, they both try to build it and one fails to start.
peek-doc-admin	peek-admin-doc	API documentation doesn't fit in anywhere, it might as well live with the admin documentation.

Table 3: Python Services

v2 Name	V3 Name	Reason
peek-server	peek-logic-service	Logic is a less ambiguous name.
peek-worker	peek-worker-service	Adding "service" clearly identifies this component as a service.
peek-agent	peek-agent-service	Adding "service" clearly identifies this component as a service.
peek-storage	peek-storage-service	Adding "service" clearly identifies this component as a service.
peek-client	peek-field-service, peek-office-service	For security reasons, we've decided to separate the backend services for Field and Office. In v2, there was limited support for specifying which app a plugin responds to. In v3, plugins can choose which service to run on.

Table 4: Core Plugins

v2 Name	V3 Name	Reason
peek-plugin-docdb	peek-core-docdb	The DocDB is required by the peek-core-search plugin, which means it also has to be a core plugin.

Table 5: Plugins

v2 Name	V3 Name	Reason
ALL peek-plugin-[pof/pon]-* plugins	peek-plugin-enmac-*	Sticking with enmac means we no longer have to manage external name changes.

PEEP7 Overview

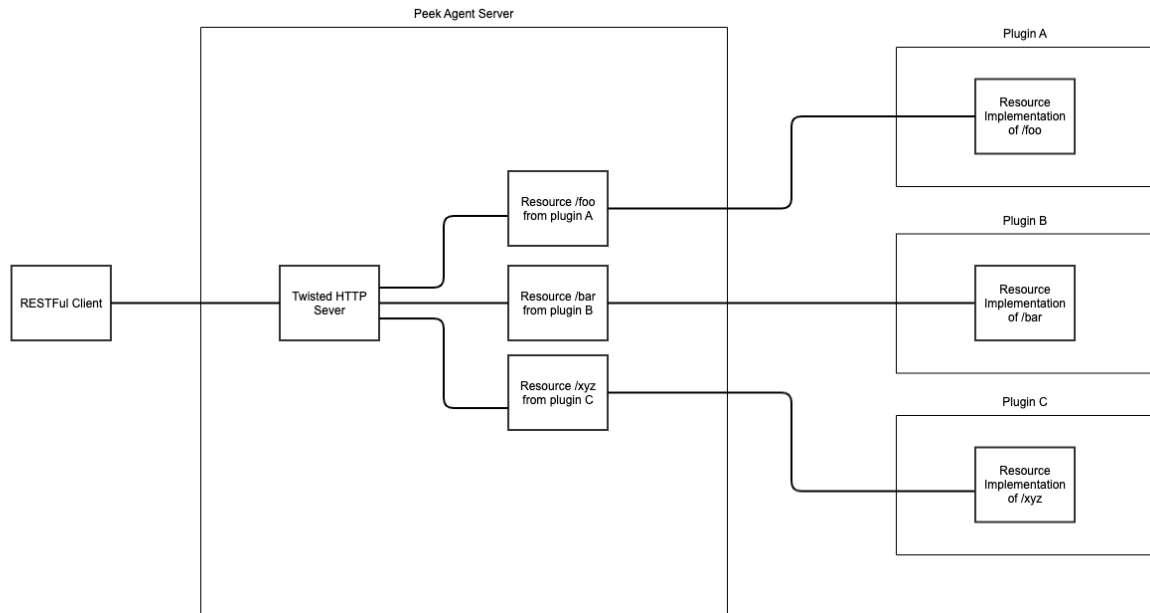
Motivation

This is designed to retrieve field assessments data by a RESTful API from an external service (works management system) where stakeholders can view field assessments assigned to a field engineer instead of fetching it from ADMS.

PEEP7: Add RESTful Service on Peek Agent Service

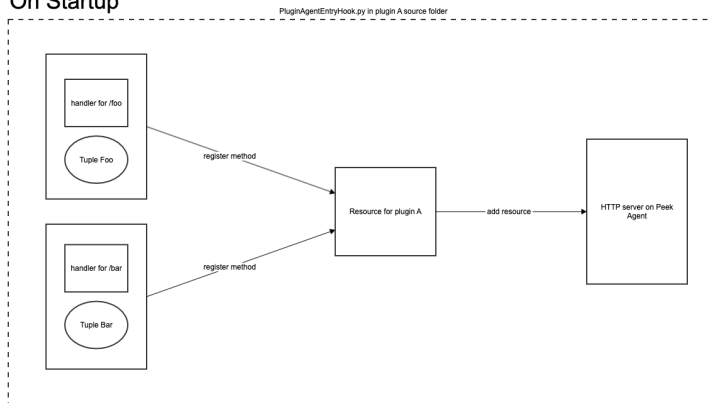
In PEEP7, RESTful Service was added to the `peek-agent-service` to allow data access via HTTP for services external to the Peek stack.

The overall architecture is explained in the diagram below:

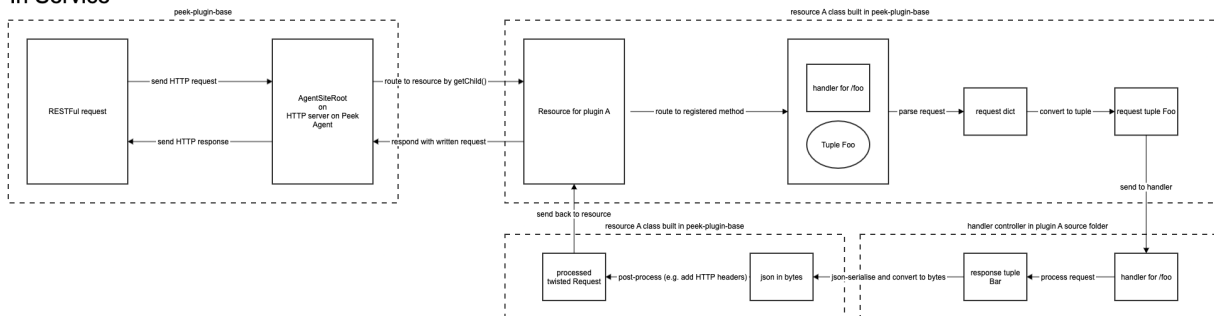


The structure of a RESTful Resource is explained in the diagram below:

On Startup



In Service



PEEP8 Overview

Motivation

As part of the project's core value of creating a highly maintainable code base, Synerty routinely upgrades the dependencies that Peek runs on.

PEEP8: Upgrade to Python 3.9.1, Node 14.15.3

As part of the project's core value of creating a highly maintainable code base, Synerty routinely upgrades the dependencies that Peek runs on.

Why update Python?

Synerty will provide the Peek v3 platform to all new upgrades and customers. Python 3.6.8 is 2 years old at this point and we don't want to go live with a new installation and old dependencies.

Python is part of the server setup steps, it's not deployed as part of the release, and it's far more difficult to roll out updates to it.

Why update Node?

Node is less of an issue as it's deployed with the release and it can be changed at any time. However, we have upgraded upgrade this dependency for the Peek v3 releases.

12.2 v3.1.x Release Notes

12.2.1 Platform Changes

Peek v3.1.0 is largely a stability and quality of life update. Few new features have been added, as the focus has been on finding and fixing any pain points or errors present after the major upgrades made in Peek 3.0.

One new feature that was added to Peek v3.1.0 is an improved API for GPS support.

12.2.2 Major Plugin Changes

We specifically avoided making any major changes to existing plugins in this release. One new plugin was created and is expected to improve an expand GPS usage in Peek significantly in future releases.

12.2.3 Deployment Changes

Deployment of the Peek platform now builds several prerequisites from source, as opposed to using a package manager, resulting in an equivalent installation across all supported platforms.

Windows Deployment

Note: This release is not supported on Windows.

Linux Deployment

Deployment for Linux remains the same after the major changes of Peek v3.0.

Debian: *Installation Guide*

Redhat: *Installation Guide*

macOS Deployment

Deployment for macOS remains the same after the major changes of Peek v3.0.

MacOS: *Installation Guide*

Windows Deployment

Nil.

Note: The windows deployment will change to use Windows Subsystem for Linux in a future release.

12.2.4 Migration Steps

Perform the following migration steps, and then restart the Peek services.

12.2.5 User Acceptance Test Results

Peek v3.1 is the most tested version of Peek to date. The results of user acceptance test cycles performed on Peek v3.1 can be downloaded below.

Download test results [here](#).

Known Issues

- **PEEK-1097 NSAT14 - 300 - Timeout Alarm - operation update** (A timeout alarm may not fire for a dispatched job in ADMS if a peek device enters flight mode while a job is being dispatched to it.)
- **PEEK-1069 OSAT05 - 450 - Comments & photos** (Combined photo-comments may not appear properly in audit logs.)
- **PEEK-1149 Devices tab not appearing in Peek DMS Search** (Devices can be searched for in the DMS diagram but cannot be selected for more detail)

12.2.6 v3.1.0 Resolved Issues Log

Bug

- PEEK-1134 Duplicate Key Login Error
- PEEK-1129 Fix Admin App DatePipe Provider Bug
- PEEK-1127 Peek office login page stays disabled after error
- PEEK-1126 Fix Peek Office Build
- PEEK-1124 Core Search: Updating search object properties to None doesn't work
- PEEK-1099 Can't Login to Peek Office
- PEEK-1098 Field search showing previous logged-in devices in result
- PEEK-1095 Field app WebSQL errors
- PEEK-1090 DocDB: Use date pipe for Date data in UI
- PEEK-1017 Fix Field Incidents Build Issue
- PEEK-919 Docdb Popup won't close on Safari browser
- PEEK-913 Field assessments type error
- PEEK-909 Move NgLifeCycleEvents from peek-plugin-base-js to vortexjs
- PEEK-908 VortexJS needs to handle logged out state
- PEEK-877 Search loses previous results when search window closes
- PEEK-866 Tooltips didn't show up and search window remain when navigate to diagram location
- PEEK-848 non-core Plugins are hard coded in peek-field-app

Task

- PEEK-1128 Replace hardcoded strings
- PEEK-1010 Complete v3.1.0 Field Switching UI
- PEEK-985 Test Peek Mobile with Peek v3.1
- PEEK-884 Cleanup old rename_plugin.sh files
- PEEK-836 Core Device - Add support for capturing GPS information
- PEEK-835 Show field devices in core-search GPS search
- PEEK-834 Position on a field device within the GIS Diagram
- PEEK-833 GIS Diagram - Show location of field units / GPS
- PEEK-798 Add Assessments Table in Field Assessment Plugin (frontend major change required)
- PEEK-769 Create Field Assessments API (non-subscription)

Improvement

- PEEK-1105 SOS Email - Add new SOS Email feature that includes the GPS
- PEEK-1047 DocDB: Add support for documents with datetime
- PEEK-783 FUI - Rewrite UI for Field Incidents
- PEEK-782 FUI - Restructure Field Incidents Navigation
- PEEK-773 FUI - Rewrite UI for Field Switching
- PEEK-772 FUI - Restructure Field Switching Navigation
- PEEK-771 PNA - Update Camera APIs
- PEEK-766 Email Incident - Google Map link to be added on Dispatch email
- PEEK-765 Email Incident - Coordinate conversion feature Easting/Northing -> Lat/Long
- PEEK-764 FAD - Include field assessment details in Email Incident details.
- PEEK-763 FAD - Include field assessment details in Field Incident details.

Sub-Task

- PEEK-1108 SOS Email - Rename peek-plugin-enmac-chat to peek-plugin-enmac-msg-sos-email
- PEEK-1107 SOS Email - Add new Email Template plugin Skeleton
- PEEK-951 FUI - Rewrite UI for Field Assessments - Photo List Screen
- PEEK-950 FUI - Rewrite UI for Field Assessments - Item Details Screen
- PEEK-949 FUI - Rewrite UI for Field Assessments - Photo Detail Screen
- PEEK-948 FUI - Rewrite UI for Field Assessments - Item List Screen
- PEEK-801 PNA - Update Camera APIs - Field Assessments
- PEEK-800 PNA - Update Camera APIs - Field Incidents
- PEEK-795 FUI - Rewrite UI for Field Incidents - Finding Detail
- PEEK-794 FUI - Rewrite UI for Field Incidents - Finding, New
- PEEK-793 FUI - Rewrite UI for Field Incidents - Call Detail
- PEEK-792 FUI - Rewrite UI for Field Incidents - Premise History List
- PEEK-791 FUI - Rewrite UI for Field Incidents - Incident Transition Dialog
- PEEK-790 FUI - Rewrite UI for Field Incidents - Incident Detail
- PEEK-789 FUI - Rewrite UI for Field Incidents - Incident Fault Report Detail
- PEEK-788 FUI - Rewrite UI for Field Incidents - Incident Third Party Detail
- PEEK-787 FUI - Rewrite UI for Field Incidents - Incident Outer Screen
- PEEK-786 FUI - Rewrite UI for Field Incidents - Finding List
- PEEK-785 FUI - Rewrite UI for Field Incidents - Call List
- PEEK-784 FUI - Rewrite UI for Field Incidents - Incident List
- PEEK-781 FUI - Rewrite UI for Field Switching - Operation List
- PEEK-780 FUI - Rewrite UI for Field Switching - Job List

- PEEK-779 FUI - Rewrite UI for Field Switching - Transition Permit
- PEEK-778 FUI - Rewrite UI for Field Switching - Transition Operation
- PEEK-777 FUI - Rewrite UI for Field Switching - Transition Job
- PEEK-776 FUI - Rewrite UI for Field Switching - Permit Details
- PEEK-775 FUI - Rewrite UI for Field Switching - Operation Details
- PEEK-774 FUI - Rewrite UI for Field Switching - Job Details

12.3 v3.0.x Release Notes

12.3.1 Platform Changes

Peek has been through a large code restructure, resulting in the renaming of many components of most plugins. Peek's UI has been massively overhauled, resulting in a much cleaner user interface. this interface update resulted in overhauled app navigation resulting in a more streamlined User experience. Updates were made to the versions of both Angular framework and and Python being used. Peek now builds native apps using capacitor.

PEEPS Executed

- PEEP4: *PEEP4 Overview*
- PEEP7: *PEEP7 Overview*
- PEEP8: *PEEP8 Overview*

12.3.2 Major Plugin Changes

All plugins were affected by a mass-renaming which also split Peek mobile into Peek office and Peek field applications. These new names are purpose-specific and platform-agnostic, better reflecting their usage.

12.3.3 Deployment Changes

Deployment of the peek platform now builds several prerequisites from source, as opposed to using a package manager, resulting in an equivalent installation across all supported platforms.

Windows Deployment

Note: This release is not supported on Windows.

Linux Deployment

Prerequisites are now built from source.

Debian: `debian_install_postgresql`

Redhat: *Install PostgreSQL*

macOS Deployment

Prerequisites are now built from source.

MacOS: *Install PostgreSQL*

iOS Deployment

Windows Deployment

Nil.

Note: The windows deployment will change to use Windows Subsystem for Linux in a future release.

12.3.4 Migration Steps

Perform the following migration steps, and then restart the Peek services.

12.3.5 v3.0.0 Issues Log

User Acceptance Test Results

Download test results [here](#).

Bug

- [PEEK-562] - Gitlab - Setup macOS builds
- [PEEK-680] - Fix login screen redirect for Peek
- [PEEK-681] - Fix back button for peek-mobile
- [PEEK-694] - 2020-Oct-Orion-UAT Fix cog in bottom right not visible on Peek Mobile.
- [PEEK-695] - 2020-Oct-Orion-UAT ORA-01805: possible error in date/time operation
- [PEEK-699] - Alembic fails on new DB - NoSuchTableError: cache_inval_hypertable
- [PEEK-701] - Add NGCC post install step to peek services
- [PEEK-710] - Search results popup only works on one result
- [PEEK-746] - Migrate Peek configuration directories v2.x to v3.x
- [PEEK-747] - Docdb popup modals don't open
- [PEEK-754] - Fix Header not appearing on page load in Peek Field App
- [PEEK-761] - Peek DMS Diagram - Symbols don't rotate
- [PEEK-829] - Fix field switching - operation detail page performance issue
- [PEEK-846] - Fix docdb popup delay issue
- [PEEK-854] - Remove all uses of ujson due to seg faulting
- [PEEK-855] - Alembic migration logs about running multiple times for each plugin it loads.

- [PEEK-857] - v2.5 DocDB Popups occur when no data is present
- [PEEK-873] - “locate” property item button doesn’t function
- [PEEK-874] - The content of exit message has a typo
- [PEEK-885] - v3 macOS build needs libgeos
- [PEEK-886] - v3 macOS build fails to install mssql
- [PEEK-919] - Docdb Popup won’t close on Safari browser

Task

- [PEEK-671] - Add OnViewInit to ComponentLifecycleEventEmitter
- [PEEK-673] - Create PEEP1: Integrate CapacitorJS Into Peek
- [PEEK-674] - Merge build-web and src folders in peek mobile, desktop & admin
- [PEEK-678] - Replace NativeScript documentation with CapacitorJS documentation
- [PEEK-679] - Add compatibility matrix to Synerty docs
- [PEEK-685] - Fix Peek field incidents photo finding
- [PEEK-688] - @angular and ng-zorro upgrade v9 to v10
- [PEEK-717] - Update Peek Packaging Scripts for MacOS and Linux
- [PEEK-718] - Segregate Peek into two sub-groups - Community & Enterprise
- [PEEK-724] - Update CI pipeline scripts to build Peek community and enterprise releases
- [PEEK-733] - Update Peek names in confluence
- [PEEK-735] - Update and demo peek_core_screen plugin
- [PEEK-738] - Update peek-field-service and peek-office-service
- [PEEK-740] - Fix issues with peek-field-switching and prepare demo
- [PEEK-742] - Remove enterprise plugin dependencies in peek-plugin-diagram-positioner
- [PEEK-743] - Add schema renaming functionality for each applicable plugin
- [PEEK-744] - Improve editing icons in peek DMS diagram
- [PEEK-745] - Rename peek plugin files, class names and imports
- [PEEK-751] - Update Peek Font Family
- [PEEK-752] - Update Peek Field and Office config page
- [PEEK-753] - Add an offline indicator to the Peek Field and Office apps
- [PEEK-755] - Update Peek Field and Office unknown route page
- [PEEK-758] - Add ant design config override file for Peek apps
- [PEEK-847] - Add The SOS Button Back
- [PEEK-853] - Create v3.0.x branch
- [PEEK-858] - Update editorconfig and format code for every project
- [PEEK-890] - v3 Update synerty-peek installs to use source for PG
- [PEEK-894] - v3 Update docs for Py 3.9.1, for macOS

- [PEEK-898] - v3 macOS, Python needs `--with-openssl --with-zlib` flags

Improvement

- [PEEK-349] - Rename peek-mobile to peek-field-app
- [PEEK-350] - Rename peek-desktop to peek-office-app
- [PEEK-353] - Split peek-client to peek-field-service and peek-office-service
- [PEEK-388] - Rename peek-server to run peek-logic-service
- [PEEK-591] - 2020-Dec Fix gitlab peek unit tests
- [PEEK-703] - Convert peek-plugin-docdb to peek-core-docdb
- [PEEK-704] - Rename all peek-plugin-pof/pon plugins to peek-plugin-enmac
- [PEEK-705] - Rename peek-worker to peek-worker-service
- [PEEK-706] - Rename peek-agent to peek-agent-service
- [PEEK-707] - Rename peek-storage to peek-storage-service
- [PEEK-709] - Configure Auto-Restart with Systemd
- [PEEK-734] - Combine GitLab Packaging Scripts and Generic Linux Packaging Script into One
- [PEEK-748] - Update VortexUtil to support multiple “accept from vortex” strings
- [PEEK-796] - PNA - Streamline the Creation of Signed Apps
- [PEEK-840] - Create peek-office-doc, rename doc-* to *-doc
- [PEEK-849] - Add platform dependency test cases
- [PEEK-889] - v3 Add support for Python 3.9.1
- [PEEK-891] - v3 Update to NODE v14.15.3
- [PEEK-892] - v3 Refactor peek-linux-sonar CI job to not need NODE_VER
- [PEEK-893] - v3 Update docs for Py 3.9.1, for Linux
- [PEEK-917] - Diagram colours printing incorrectly

Sub-Task

- [PEEK-687] - Upgrade ant design to v10
- [PEEK-689] - Upgrade angular to v10
- [PEEK-690] - Create PEEP2 Document

12.4 v2.5.x Release Notes

12.4.1 Platform Changes

The peek-office-service and peek-logic-service now support a PEM bundle file for SSL.

The peek desktop, admin & mobile web-apps have been upgraded to use Angular 9.

The following Synerty projects have been deprecated:

- ng2-balloon-msg, ng2-balloon-msg-web, ng2-balloon-msg-ns
- peek-theme, peek-field-app-theme-test
- peek-util, peek-util-web, peek-util-ns
- font-awesome
- nativescript-peek-update

12.4.2 Major Plugin Changes

The following plugins have been developed for the v2.5.x+ Peek releases.

- **peek-plugin-base-js** (Open Source) This plugin contains shared code that all Peek projects can access and utilise. The following Synerty projects have been merged into this plugin:
 - ng2-balloon-msg, ng2-balloon-msg-web
 - peek-theme
 - peek-util, peek-util-web

12.4.3 Deployment Changes

Windows Deployment

Note: This release is not supported on Windows.

Linux Deployment

Nil

macOS Deployment

Nil

iOS Deployment

Note: Peek v2.5.x does not have support for iOS, this will be updated in the v3.0.x release. We're going to use Ionics Capacitor framework to create a full hybrid app.

Android Deployment

Note: Peek v2.5.x does not have support for Android, this will be updated in the v3.0.x release. We're going to use Ionics Capacitor framework to create a full hybrid app.

12.4.4 Migration Steps

In PEEK-660, the peek-office-service and peek-logic-service projects now accept a PEM bundle file to serve SSL. Follow the steps in the link below to set this up.

[Peek Client & Server SSL](#)

12.4.5 v2.5.4 Issues Log

Bug

- [PEEK-761] - Peek DMS Diagram - Symbols don't rotate

Improvement

- [PEEK-917] - Diagram colours printing incorrectly

12.4.6 v2.5.3 Issues Log

Bug

- [PEEK-680] - Fix login screen redirect for Peek
- [PEEK-681] - Fix back button for peek-mobile
- [PEEK-710] - Search results popup only works on one result
- [PEEK-761] - Peek DMS Diagram - Symbols don't rotate
- [PEEK-829] - Fix field switching - operation detail page performance issue
- [PEEK-846] - Fix docdb popup delay issue
- [PEEK-854] - Remove all uses of ujson due to seg faulting
- [PEEK-866] - Tooltips didn't show up and search window remain when navigate to diagram location
- [PEEK-869] - Fix Popup Fade Issue
- [PEEK-873] - "locate" property item button doesn't function
- [PEEK-874] - The content of exit message has a typo

12.4.7 v2.5.2 Issues Log

Bug

- [PEEK-700] - DMS diagram on-click tooltip missing

Improvement

- [PEEK-709] - Configure Auto-Restart with Systemd

12.4.8 v2.5.1 Issues Log

Bug

- [PEEK-682] - Can't find stylesheet '~@synerty/peek-plugin-base-js/src/styles/ant-design'
- [PEEK-711] - PEEK_RELEASE gitlab build fails, "requirement sphinx"

12.4.9 v2.5.0 Issues Log

Bug

- [PEEK-504] - Toggle feeder colours - breakers go black
- [PEEK-618] - synerty-peek pip_uninstall_and_develop.sh
- [PEEK-663] - NorthPower - Peek, Equipment popup pops up off the screen.
- [PEEK-664] - synerty-peek, fix typo "hypons"
- [PEEK-670] - Fix Balloon messages not appearing
- [PEEK-672] - Fix safari issue with Ant Design tabs disappearing
- [PEEK-667] - AUTH - Add a workaround for Peek-Admin on Safari with Angular9

New Feature

- [PEEK-660] - txHttpUtil Peek doesn't serve SSL certificates correctly

Task

- [PEEK-668] - Create Peek Release v2.5.0

Improvement

- [PEEK-632] - Upgrade peek UI ng-zorro v9
- [PEEK-639] - @angular, ng-zorro, ng-zorro-mobile Upgrade 8.1 to 9.1
- [PEEK-264] - Ensure route.params is unsubscribed.

12.5 v2.4.x Release Notes

12.5.1 Platform Changes

VortexPY and VortexJS have been updated to allow datetimes in TupleSelector classes.

The platform now requires the "timescale" PostgreSQL extension.

12.5.2 Major Plugin Changes

The following plugins have been developed for the v2.4.x+ Peek releases.

- **peek-plugin-eventdb** (Open Source) This plugin stores current and historical alarms and events.
- **peek-plugin-enmac-event-loader** (Proprietary) This plugin loads historical and current alarms and events from the ENMAC.
- **peek-core-search** (Open Source) The search feature of Peek has been updated to support faster partial keyword searches.
- **peek-plugin-enmac-equipment-loader** (Proprietary) The Equipment Loader plugin now has a list of COMPONENT_CLASSES that determines which equipment will be loaded into the search for indexing.

12.5.3 Deployment Changes

Windows Deployment

Note: This release is not supported on Windows.

Linux Deployment

Reinstall PostgreSQL as per the updated instructions to include support for timescale.

Debian: `debian_install_postgresql`

Redhat: *Install PostgreSQL*

macOS Deployment

Reinstall PostgreSQL as per the updated instructions to include support for timescale.

MacOS: *Install PostgreSQL*

iOS Deployment

Note: Peek v2.0.x does not have support for iOS, this will be updated in a future release. We're going to Ionics Capacitor framework to create a full hybrid app.

Windows Deployment

Nil.

Note: The windows deployment will change to use Windows Subsystem for Linux in a future release.

12.5.4 Migration Steps

Perform the following migration steps, and then restart the Peek services.

peek-core-search

This update requires reloading all of the search data. Run the following to truncate the search data.

```
psql <<EOF
-- Clear out and reload all the Search data

TRUNCATE TABLE core_search."EncodedSearchIndexChunk" CASCADE;
TRUNCATE TABLE core_search."SearchIndex" CASCADE;
TRUNCATE TABLE core_search."SearchIndexCompilerQueue" CASCADE;

TRUNCATE TABLE core_search."EncodedSearchObjectChunk" CASCADE;
TRUNCATE TABLE core_search."SearchObject" CASCADE;
TRUNCATE TABLE core_search."SearchObjectCompilerQueue" CASCADE;

EOF
```

peek-plugin-enmac-equipment-loader

The equipment loader will need to reload the search data, force an update with the following SQL.

```
psql <<EOF
UPDATE pl_enmac_equipment_loader."ChunkLoadState"
SET "lastSearchHash" = 'reloadme';
EOF
```

peek-plugin-enmac-switching-loader

The equipment loader will need to reload the search data, force an update with the following SQL.

```
psql <<EOF
UPDATE pl_enmac_switching_loader."ChunkLoadState"
SET "lastSearchHash" = 'reloadme';
EOF
```

peek-plugin-eventdb

Enable this plugin all Peek services `config.json`, in the enabled plugins. This must be after the DocDB plugin.

```
peek-plugin-eventdb
```

peek-plugin-enmac-event-loader

Enable this plugin all Peek services `config.json`, in the enabled plugins. This must be after the EventDB plugin.

```
peek-plugin-enmac-event-loader
```


12.5.5 v2.4.5 Issues Log

Bug

- [PEEK-650] - Alarm/Event - Admin, Office and Field angular compiles

12.5.6 v2.4.4 Issues Log

Bug

- [PEEK-597] - **Diagram: Deleted grids are not removed from browser cache**, Deleted PowerOn pages / overlays are not removed from the diagram
- [PEEK-604] - Diagram flashes when PowerOn updates pages
- [PEEK-650] - Alarm/Event - Bugfixedm Add switch for “Alarms Only” switch
- [PEEK-653] - **Alarms/Events - Alarm Zone default property values are wrong**. They now load from the correct lookup.

12.5.7 v2.4.3 Issues Log

Bug

- [PEEK-649] - Alarms/Events - ShowOnPopup won't disable from admin property editor

Task

- [PEEK-648] - Alarms/Events - Include Component Alias in Alarm Data
- [PEEK-650] - Alarms/Events - Added switch for “Alarms Only” next to “Live” switch

12.5.8 v2.4.2 Issues Log

Bug

- [PEEK-643] - EventDB - Event date times are loaded without timezone
- [PEEK-645] - Event WebView Loader - Fixed last loaded date timezone

Task

- [PEEK-646] - Release v2.4.2

CI/CD Tasks

- [PEEK-644] - Gitlab - Update Gitlab sonar Docker tag

12.5.9 v2.4.1 Issues Log

Bug

- [PEEK-635] - Core Search - Add support for searching for two letter words.
- [PEEK-636] - **Plugin EventDB - TupleSelectors arn't supporting datetimes**, this was caused by a missing merge in peek-office-app.
- [PEEK-637] - EventDB PoN Loader - Finalise property loading, and set to load once.

Task

- [PEEK-638] - Release v2.4.1

12.5.10 v2.4.0 Issues Log

Bug

- [PEEK-455] - Peek Platform - Using overlay directories causes frontend to always rebuild
- [PEEK-510] - Starting an edit of a branch should re-position the diagram.
- [PEEK-621] - Diagram - Polygons fill outside of boxes with dynamics, (when it's over 100%)
- [PEEK-622] - ENMAC GraphDB Loader: 'NoneType' object has no attribute 'updateInProgressDate'
- [PEEK-630] - DMS Diagram - Font alignments for middle and right don't work
- [PEEK-631] - DMS Diagram - Bold Helvetica doesn't render correctly
- [PEEK-634] - Peek DMS Diagram - Layer Min/Max Display Ranges Overlap

New Feature

- [PEEK-522] - Develop Alarms and Events viewer

Task

- [PEEK-623] - Release v2.4.0, including release notes

Improvement

- [PEEK-608] - Prevent peek from loading the same plugin twice.
- [PEEK-609] - ENMAC Equipment Loader - Filter components added to search based on component id
- [PEEK-610] - Search - Update tokenizing to allow partial keyword support
- [PEEK-612] - Add platform support for Timescale PostgreSQL
- [PEEK-614] - Add adaptor/patching for running worker tasks in plpython
- [PEEK-615] - Abstract Index - Run deduplicate SQL before fetching more blocks.
- [PEEK-626] - Storage - runPyInPg use tuples instead of plain json for args and returns

- [PEEK-627] - Vortex - Add support for TupleSelectorUpdateMapper in TupleDataObservable
- [PEEK-629] - txhttputil - Add support for url args in HttpResourceProxy

CI/CD Tasks

- [PEEK-605] - Gitlab - error: The ‘sphinx’ distribution was not found and is required by sphinx-rtd-theme, peek-doc-dev
- [PEEK-606] - Gitlab - Unit tests can’t find non open source peek plugins (peek-plugin-enmac-*)

12.6 v2.3.x Release Notes

12.6.1 Platform Changes

Nil

12.6.2 Plugin Changes

The following plugins have code changes on the v2.3.x release branch

- peek-plugin-diagram
- peek-plugin-enmac-diagram-loader

12.6.3 Deployment Changes

Windows Deployment

Note: This release is not supported on Windows.

Linux Deployment

Nil

macOS Deployment

Nil

iOS Deployment

Note: Peek v2.0.x does not have support for iOS, this will be updated in a future release. We’re going to Ionics Capacitor framework to create a full hybrid app.

Windows Deployment

Nil.

Note: The windows deployment will change to use Windows Subsystem for Linux in a future release.

12.6.4 Migration Steps

peek-plugin-enmac-diagram-loader

In PEEK-526, the “Layer Profile Name” has been renamed to “Environment Name” in the plugin settings.

The migration steps for this change are as follows:

- Login to Peek Admin,
- Go to plugin “ENMAC Diagram Loader”
- Go to the “Edit App Server Settings” tab.
- Update “Environment Name” to a value from “ENMAC”.“ENVIRONMENT_LIST”.“ENVIRONMENT_NAME” that you wish the diagram to load with.
- Click Save
- Run the following as peek on the peek server to force a reload of the data:

```
psql <<EOF
UPDATE pl_enmac_diagram_loader."PageLoadState"
SET  "lastDisplayHash"='reloadme';
EOF
```

- Restart the peek agent.

```
sudo systemctl restart peek_agent_service
```

12.6.5 v2.3.3 Issues Log

Bug

- [PEEK-455] - Peek Platform - Using overlay directories causes frontend to always rebuild
- [PEEK-501] - Lines and circles are imported from PowerOn with width=0. These lines were displayed with an unscaled 1pt line.
- [PEEK-505] - Branches are listed twice
- [PEEK-607] - Queue Compilers get held up too much by fetching duplicates.

Improvement

- [PEEK-608] - Prevent peek from loading the same plugin twice.
- [PEEK-614] - Add adaptor/patching for running worker tasks in plpython

12.6.6 v2.3.2 Issues Log

This was an internal release.

12.6.7 v2.3.1 Issues Log

Bug

- [PEEK-526] - Loading Poke Points was buggy, loaded two disps with the same action and didn't gracefully handle multiple poke points to the same viewport from the same page.

12.6.8 v2.3.0 Issues Log

Bug

- [PEEK-602] - ENMAC Diagram Loader - Agent no longer reloads all pages on restart.

New Feature

- [PEEK-526] - Implement support for loading in ENMAC "Poke Points". These are hotspots that jump the user to another location on the diagram when clicked.

CI/CD Tasks

- [PEEK-601] - Setup Peek release builds to pin Docker and Python dependencies for the life of that release branch. (EG v2.2.7 will have the exact same python dependency packages, Twisted, SQLAlchemy, VortexPY, etc as v2.2.0)

12.7 v2.2.x Release Notes

12.7.1 Platform Changes

1. PostgreSQL upgrade from v10 to v12
2. The peek platform now requires the postgresql-plpython3 package installed. (See Migration Steps)
3. A new peek service has been added "peek-storage-service". This currently doesn't run as its own process but it will in the future.

12.7.2 Plugin Changes

There has been no functional changes to Peek in this release. There are a lot of performance improvements in this release of peek, the following plugins have changed quite significantly.

- peek-plugin-diagram
- peek-core-search
- peek-core-docdb
- peek-plugin-graphdb

- peek-plugin-livedb
- peek-plugin-index-blueprint

12.7.3 Deployment Changes

Windows Deployment

Note: This release is not supported on Windows.

Linux Deployment

- Upgrade Postgresql-10 to Postgresql-12
- Installing PlPython3u Postgresql extension

As this is the typical production install, there are migration notes below.

macOS Deployment

- Upgrade Postgresql-10 to Postgresql-12
- Installing PlPython3u Postgresql extension

See the *Setup OS Requirements macOS* documentation for the updates, the following sections need to be completed again. “Install PostgreSQL” and “Install Python 3.6”.

iOS Deployment

Note: Peek v2.0.x does not have support for iOS, this will be updated in a future release. We’re going to Ionics Capacitor framework to create a full hybrid app.

Windows Deployment

Nil.

Note: The windows deployment will change to use Windows Subsystem for Linux in a future release.

12.7.4 Migration Steps

Redhat 7.5+

Follow the migration steps to complete the upgrade of Postgresql-10 to Postgresql-12 and install the plpython3 package.

Stop Peek

Start the migration tasks with Peek stopped.

Stop peek with the following commands

```
sudo true
sudo systemctl stop peek_agent_service
sudo systemctl stop peek_office_service
sudo systemctl stop peek_worker_service
sudo systemctl stop peek_logic_service
```

Upgrade to PostgreSQL-12

Install PostgreSQL 12

```
URL="https://download.postgresql.org/pub/repos/yum"
URL=$URL/reporepms/EL-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm"
sudo yum install -y $URL

PKG="postgresql12"
PKG="$PKG postgresql12-server"
PKG="$PKG postgresql12-contrib"
PKG="$PKG postgresql12-devel"
PKG="$PKG postgresql12-plpython3"
sudo yum install -y $PKG
```

Alternatively, the RPMs can be downloaded from here and manually installed.

https://download.postgresql.org/pub/repos/yum/12/redhat/rhel-7.7-x86_64/postgresql12-contrib-12.2-2PGDG.rhel7.x86_64.rpm

https://download.postgresql.org/pub/repos/yum/12/redhat/rhel-7.7-x86_64/postgresql12-libs-12.2-2PGDG.rhel7.x86_64.rpm

https://download.postgresql.org/pub/repos/yum/12/redhat/rhel-7.7-x86_64/postgresql12-plpython3-12.2-2PGDG.rhel7.x86_64.rpm

https://download.postgresql.org/pub/repos/yum/12/redhat/rhel-7.7-x86_64/postgresql12-devel-12.2-2PGDG.rhel7.x86_64.rpm

https://download.postgresql.org/pub/repos/yum/12/redhat/rhel-7.7-x86_64/postgresql12-server-12.2-2PGDG.rhel7.x86_64.rpm

https://download.postgresql.org/pub/repos/yum/12/redhat/rhel-7.7-x86_64/postgresql12-12.2-2PGDG.rhel7.x86_64.rpm

These can then be installed with

```
cd where_you_put_the_rpms
yum install -u *.rpm
```

The existing postgresql10 server needs to be present, if it's not reinstall it with the following command

```
sudo yum install postgresql10-server
```

Vacuum the DB so it's smaller for the migration

```
sudo systemctl start postgresql-10
sudo -u peek time echo "VACUUM FULL FREEZE;" | psql
```

Ensure the services are stopped

```
sudo systemctl stop postgresql-10 || true
sudo systemctl stop postgresql-12 || true
```

Initialise the new v12 cluster

```
sudo /usr/pgsql-12/bin/postgresql-12-setup initdb
```

Migrate old cluster to the new cluster

```
sudo su - postgres
export PGDATAOLD=/var/lib/pgsql/10/data
export PGDATAOLD=/var/lib/pgsql/12/data
export PGBINOLD=/usr/pgsql-10/bin
export PGBINNEW=/usr/pgsql-12/bin
cd /var/lib/pgsql
${PGBINNEW}/pg_upgrade
exit
```

Remove the v10 software

```
sudo yum remove postgresql10*
```

Ensure the pg_hba is setup

```
F="/var/lib/pgsql/12/data/pg_hba.conf"
if ! sudo grep -q 'peek' $F; then
    echo "host    peek    peek    127.0.0.1/32    trust" | sudo tee $F -a
    sudo sed -i 's,127.0.0.1/32    ident,127.0.0.1/32    md5,g' $F
fi
```

Enable and start postgresql 12

```
sudo systemctl enable postgresql-12
sudo systemctl start postgresql-12
```

Analyse the new cluster, the migration doesn't bring across planning statistics.


```
sudo su - postgres -c time /var/lib/pgsql/analyze_new_cluster.sh
```

Delete the old cluster

```
sudo su - postgres -c time /var/lib/pgsql/delete_old_cluster.sh
```

Grant PostgreSQL Peek Permissions

The PostgreSQL server now runs parts of peeks python code inside the postgres/postmaster processes. To do this the postgres user needs access to peeks home directory where the peek software is installed.

Grant permissions

```
sudo chmod g+rx ~peek
sudo usermod -G peek postgres
```

Restart Peek

Restart all Peek services

```
sudo true
sudo systemctl start peek_logic_service
sudo systemctl start peek_worker_service
sudo systemctl start peek_agent_service
sudo systemctl start peek_office_service
```

12.7.5 v2.2.2 Issues Log

Bug

- [PEEK-602] - ENMAC Diagram Loader - Agent no longer reloads all pages on restart.

12.7.6 v2.2.1 Issues Log

This release was required to resolve a tag and publish release issue PEEK-539

12.7.7 v2.2.0 Issues Log

Bug

- [PEEK-541] - SequenceGeneratorLimitExceeded “DispCompilerQueue_id_seq” (2147483647)
- [PEEK-550] - Peek LiveDB updates accumulate in memory and crashes
- [PEEK-552] - Python base64.py bug causes memory leak
- [PEEK-555] - Peek server doesn't shutdown while celery is waiting for a task result

- [PEEK-556] - Queue processors chew a lot of CPU when it's paused for duplicates
- [PEEK-557] - Queue Compilers skip some queue items and never compile them
- [PEEK-560] - Chunked Indexes are slow because they use the ORM to query and send to client
- [PEEK-571] - Implement Agent Sequential Loading
- [PEEK-572] - Agent loaders fail to requeue failed items
- [PEEK-578] - ENMAC Event - Loader fails to load if DB isn't initialised
- [PEEK-588] - VortexPayloadProtocol has concurrent calls to `_processData`
- [PEEK-593] - ujson crashes peek when decoding doubles from realtime loader
- [PEEK-595] - Redis connection interruptions cause memory leak
- [PEEK-567] - VortexPY - Add PushProducer to VortexServerConnection
- [PEEK-568] - VortexPY - Add support for sendVortexMsg priority

New Feature

- [PEEK-179] - Storage Service

Improvement

- [PEEK-589] - Add code to queue compiler to periodically vacuum the queue, index and encoded data tables.
- [PEEK-531] - Add config.json settings for max XXX per child settings
- [PEEK-377] - Add config.json settings for sending service logs to syslog
- [PEEK-581] - Add config.json settings for logs kept and log size
- [PEEK-527] - Add config.json settings for memory debug dumping
- [PEEK-554] - Add config.json settings for twisted celery max connection times
- [PEEK-553] - Upgrade Twisted to latest version (remove pin from <19.0.0)
- [PEEK-559] - Refactor queue compilers and client handlers to use Abstract Chunked Index
- [PEEK-561] - Platform - Restart services without the unbuffered flag (performance)
- [PEEK-590] - Create `peek_abstract_chunked_data_loader` plugin
- [PEEK-574] - Refactor queue compiler client CacheController to inherit Abstract Chunked Index
- [PEEK-575] - Update LiveDB Realtime value loader to use the `plpython3u` Tuple load methods
- [PEEK-576] - Update the Abstract Chunk Index client loader to use `plython3u` Tuple load methods
- [PEEK-577] - Update the Abstract Chunked Index Queue controllers to use `plpython3u` to load their blocks.
- [PEEK-563] - Add `plpython3u` support to PostgreSQL to documentation
- [PEEK-583] - Peek Storage - Create `runPyInPg` method to run any function in PostgreSQL
- [PEEK-586] - Add `TcpVortex` memory leak unit tests to VortexPY
- [PEEK-596] - User Plugin : Add option for accepting invalid LDAP SSL certificate
- [PEEK-599] - ENMAC Realtime Loader - Add settings for background polling in admin UI

CI/CD Tasks

In this release, Synerty has rebuilt our CI/CD build servers to use gitlab and gitlab pipelines.

- [PEEK-538] - Make pipeline builds work with merge commits.
- [PEEK-539] - Tutorial Documentation and Example plugin: Peek Versioning inconsistency
- [PEEK-566] - Branch builds of a peek-release now checkout all plugins on the same branch.

12.8 v2.1.x Release Notes

12.8.1 Platform Changes

Peek Worker

We've experienced issues with the Celery library that Peek uses for distributed processing. The issue observed is Redis closing client connections due to a Pub/Sub buffer overflow. This is believed to be due to Celery not unsubscribing from events properly, however investigations showed that the result keys are being cleaned up in Redis. We're not sure what the fix is, Hence the passive language of this paragraph.

What we have done is improved the txCelery-Py located at <https://github.com/Synerty/txcelery-py3/commits/master> :

- Task create and result capture are performed in the same thread.
- Tasks are automatically retried if they fail with a redis connection error.
- **Performance has been greatly reduced by staying in the thread and adding a timeout** in the thread. There was a significant overhead in Twisted from entering and exiting threads.

These changes actually all applied to the Peek Server, but they were for distributed tasks.

12.8.2 Plugin Changes

Nil

12.8.3 Deployment Changes

Linux Deployment

Nil

macOS Deployment

Nil

iOS Deployment

Note: Peek v2.0.x does not have support for iOS, this will be updated in a future release. We're going to Ionics Capacitor framework to create a full hybrid app.

Windows Deployment

Nil.

Note: The windows deployment will change to use Windows Subsystem for Linux in a future release.

12.8.4 Migration Steps

Follow the following migration steps to rebuild the data that has changed in this update.

Stop Peek

Start the migration tasks with Peek stopped.

On Linux this can be done with

```
# Stop Peek
sudo true
sudo systemctl stop peek_agent_service
sudo systemctl stop peek_office_service
sudo systemctl stop peek_worker_service
sudo systemctl stop peek_logic_service
```

Clear Redis and RabbitMQ Queues

Run the following commands to clear any queues

```
# Purge any outstanding results from the redis task result store
sudo redis-cli flushall

# Purge the RabbitMQ Task Queue
sudo rabbitmqctl purge_queue celery
```

Restart Peek

Restart all Peek services

```
sudo true
sudo systemctl start peek_logic_service
sudo systemctl start peek_worker_service
sudo systemctl start peek_agent_service
sudo systemctl start peek_office_service
```

12.8.5 v2.1.7 Issues Log

This release was used to test Synertys new CI-CD builds.

12.8.6 v2.1.6 Issues Log

This release was used to test Synertys new CI-CD builds.

12.8.7 v2.1.5 Issues Log

Bug

- [PEEK-514] - Peek runs out of memory and celery connections die
- [PEEK-525] - Peek Worker chews all ram and crashes
- [PEEK-528] - Running the Peek Agent causing netservd errors on the connected PowerOn Server:
- [PEEK-531] - Add settings for max XXX per child settings

Improvement

- [PEEK-532] - Update deploy_platform_xxxx.sh/ps1 to deploy_release_xxxx.sh/ps and include installing the plugins

12.8.8 v2.1.4 Issues Log

This release was used to test Synertys new CI-CD builds.

12.8.9 v2.1.3 Issues Log

This release was used to test Synertys new CI-CD builds.

12.8.10 v2.1.2 Issues Log

Bug

- [PEEK-494] - **PowerOn Diagram Loader - RpcForAgentDisplImport.storeStateInfoTuple** unique constraint violation
- [PEEK-499] - Field crews using Peek have null token issues on iOS Safari
- [PEEK-500] - TupleStorageIndexedDbService saveTuplesEncoded never resolves promise
- [PEEK-503] - Allow auto enroll - remove device register
- [PEEK-506] - Alphabetically order Symbol Selection
- [PEEK-512] - txcelery local variable 'async_result' referenced before assignment
- [PEEK-513] - **VortexJS - IndexedDB OfflineTupleActonService** throws an error every check
- [PEEK-514] - Peek runs out of memory and celery connections die.
- [PEEK-516] - NAR ID including user and date incorrect in E-mail.

Improvement

- [PEEK-493] - Peek Platform - Add stop_peek.sh script
- [PEEK-508] - Remove default text from placed symbols.

12.8.11 v2.1.1 Issues Log

Bug

- **[PEEK-456] - Core User - Fix user title constraint for users logging in from two ADs** with different user-name
- [PEEK-482] - Diagram Edit - Order the edit branches list in descending order.
- **[PEEK-483] - Diagram Edit - Set a minimum width for the Ant.Design dropdowns** in the shape properties.
- [PEEK-488] - Queue Compilers - Fail to retry failing task, they wait indefinitely
- [PEEK-489] - Peek Platform - Twisted thread count is far too small.
- [PEEK-490] - Queue Compilers - Use too much CPU when not doing anything but waiting
- [PEEK-491] - Peek server keeps crashing with redis pub/sub buffer overflows.
- **[PEEK-492] - PoN Diagram Loader - Too many concurrent SSH connections during load** causes SSH rate limiting

12.8.12 v2.1.0 Issues Log

New Feature

- [PEEK-467] - Diagram Edit - Insert Edge components/templates/symbols

Improvement

- [PEEK-466] - Diagram Edit - Edit placed component text

Bug

- [PEEK-320] - ENMAC Graph DB Loader - Loader continually reimports trace configs
- [PEEK-457] - Core User / Device - User login sticks on login screen
- [PEEK-462] - Core Login - login screen hangs after selecting the login button
- [PEEK-463] - ENMAC Switching - program details not shown sequentially
- [PEEK-464] - Diagram Edit - Deleting existing display items doesn't work
- [PEEK-465] - Diagram Edit - Creating text doesn't popup shape properties
- [PEEK-468] - ENMAC SOAP - SOAP isn't compatible with PowerOn Advantage
- **[PEEK-469] - Field Switching - Field confirm for WEB doesn't work**
 - Time date is out of range

- [PEEK-470] - ENMAC SOAP - Datetimes now seem to be double localised
- **[PEEK-473] - All Loaders - Workers failing all their retries can** end up with queue items in limbo
- [PEEK-474] - Diagram - Locating on key only (no coord set), will fail
- [PEEK-475] - ENMAC Equipment Loader - Loader no longer loads conductors
- [PEEK-476] - Worker - Fix retry issues with redis connections
- [PEEK-477] - Queue Compilers - Ensure a chunk isn't compiled twice in paralleled
- [PEEK-478] - Loader Plugins - Fix worker retry/complete logging messages
- [PEEK-479] - Diagram - Improve LiveDB item create / poll sequence
- [PEEK-480] - Diagram - Force Lookup imports to be run sequentially
- [PEEK-481] - VortexPY - Fix reporting of blocking endpoints

12.9 v2.0.x Release Notes

12.9.1 Platform Changes

Platform Backend

The Peek Admin UI now requires a login.

The admin login recovery password can be found in `~/peek-logic-service.home/config.json`.
Under the path `httpServer.admin.user`.

The Peek-Client and Peek-Admin services now provide SSL support, see their respective `config.json` file, just provide valid files and SSL will be enabled.

The Peek-Client and Peek-Server services have combined the Web-App, websocket and Docs HTTP servers into the one HTTP port.

Docs are available at <http://peekserver:port/docs>

Platform Frontend

This release has upgraded the following major libraries

- Angular from v6.x to v8.x
- Typescript from v2.7.x to v3.4.x
- Other smaller npmjs dependencies, such as rxjs
- An introduction of ng.ant.design, Bootstrap will be phased out.

LDAP Authentication

LDAP Authentication has been improved, it now support multiple LDAP configs, including multiple LDAP servers, or use it just for different combinations of OUs/Groups.

LDAP Configuration settings must be migrated manually

Peek Desktop

The Peek Desktop has had a face lift, the unsightly side bar has been slimmed down and neatened up.

The scroll bars should not appear any longer.

12.9.2 Plugin Changes

DocDB Plugin

The DocDB plugin has new information popups:

- Tooltip
- Summary
- Details

These popups are triggered via the DocDB APIs from other plugins.

Search Plugin

The Peek search has been revamped:

- It now shows as a modal.
- The search results have been cleaned up
- The search results now show actions from the DocDB plugin popups

Diagram Generic Menu

This plugin has been renamed from “peek_plugin_generic_diagram_menu” to peek_plugin_docdb_generic_menu” as it now operates on the new DocDB popups.

Diagram

The diagram plugin has the following changes.

VIEW Updates:

- The top left button now lets users select the Coord Set
- The draw that used to load on the right hand side is now gone. The diagram instead triggers tooltip and summary popups from DocDB.

EDIT Updates:

- Edit support now highlights a templates as just one object.
- Edit support now lets you rotate a templates
- New templates placed on the diagram are no longer invisible.

Diagram Positioner

The old `peek_plugin_gis_dms_positioner` plugin has been resurrected. This plugin now uses the DocDB popups, shows locate on actions using the coord-set names, and is smart enough to not show locate actions for the current coord-set.

This replaces the old method of just using the result sof peek-core-search.

Diagram Loaders

All diagram loaders no longer load positions into the search db.

12.9.3 Deployment Changes

Linux Deployment

Nil

macOS Deployment

Nil

iOS Deployment

Peek v2.0.x does not have support for iOS, this will be updated in a future release

Windows Deployment

Nil.

Note: The windows deployment will change to use Windows Subsystem for Linux in a future release.

12.9.4 Migration Steps

Follow the following migration steps to rebuild the data that has changed in this update.

Stop Peek

Start the migration tasks with Peek stopped.

On Linux this can be done with

```
# Stop Peek
sudo true
sudo systemctl stop peek_agent_service
sudo systemctl stop peek_office_service
sudo systemctl stop peek_worker_service
sudo systemctl stop peek_logic_service
```

Redis Conf Update

The Redis Publisher/Subscriber buffer overflows and causes the task to fail, and the agent to retry. (See PEEK-317)

Double the buffer size with the following script

```
# Prime SUDO
sudo true

OLD="client-output-buffer-limit pubsub 32mb 8mb 60"
NEW="client-output-buffer-limit pubsub 64mb 16mb 90"
F="/etc/redis.conf"

# Check what it is now
grep pubsub $F

# Increase the size
sudo sed -i "s/${OLD}/${NEW}/g" $F

# Check that the change worked
grep pubsub $F

# Restart Redis
sudo systemctl restart redis
```

Enable New Plugins

Update the peek config.json files.

1. Edit each of C:\Users\peek-XXXX\homeconfig.json
2. Add `peek_plugin_diagram_positioner` just after `peek_plugin_diagram_trace`
3. Add `peek_plugin_enmac_switching_loader` just after `peek_plugin_enmac_equipment_loader`
4. Rename `peek_plugin_diagram_generic_menu` to `peek_plugin_docdb_generic_menu` or on Linux:

```
sed -i 's/_diagram_generic_menu/_docdb_generic_menu/g' ~/peek*.home/config.
↪ json
```

Truncate Load States

Due to the changes to the search and diagram plugins, all of the data will need to be reloaded.

Run the following SQL, for each applicable plugin that you have installed.

```
psql <<EOF

-- Search Plugin
DROP SCHEMA core_search CASCADE;

-- DocDB Plugin
DROP SCHEMA pl_docdb CASCADE;

-- ENMAC Equipment Loader
```

(continues on next page)

(continued from previous page)

```

TRUNCATE TABLE pl_enmac_equipment_loader."ChunkLoadState";

-- ENMAC Diagram Loader
DELETE FROM pl_enmac_diagram_loader."PageLoadState"
WHERE "scope" = 'normal';

-- ENMAC Pof GIS Location loader
TRUNCATE TABLE pl_enmac_gis_location_loader."ChunkLoadState";

-- GraphDB
TRUNCATE TABLE pl_graphdb."GraphDbChunkQueue";
TRUNCATE TABLE pl_graphdb."GraphDbEncodedChunk";
TRUNCATE TABLE pl_graphdb."GraphDbSegment";

TRUNCATE TABLE pl_graphdb."ItemKeyIndexCompilerQueue";
TRUNCATE TABLE pl_graphdb."ItemKeyIndexEncodedChunk";
TRUNCATE TABLE pl_graphdb."ItemKeyIndex";

-- ENMAC GraphDB Loader
TRUNCATE TABLE pl_enmac_graphdb_loader."GraphSegmentLoadState";

EOF

```

Start up the Peek Server manually, it will:

- Rebuild the admin site
- Migrate the database

Open a shell or command prompt and run `run_peek_logic_service`

Once the Peek Server has finished loading :

1. Load up the peek-admin-app screen at <http://<peek-logic-service>:port>.
2. Click the “Login” button, this will cause the server to write a recovery user to the `peek-logic-service.home/config.json` file with a random password.
3. Use this recovery username and password to login to the Peek Admin UI.

Once Peek Server has finished running, kill it with CTRL+C

Restart Peek

Restart all Peek services.

For windows, restart the `peek-logic-service` service then start the `peek-restarter` service, the agent, worker and client will now start.

Reconfigure LDAP

Reconfigure the new LDAP settings from the Peek-Admin site, under Platform -> Users

1. Enable the use of LDAP from the **General Settings** tab.
2. Configure the new LDAP settings using the new LDAP Settings tab.

Reconfigure Search Properties

Reconfigure the search properties from the Peek-Admin site, under Platform -> Search

Peek Admin Dashboard
Platform
Plugins
Product of Synerty Pty Ltd, All Rights Reserved
Version ###PEEKVER###

Home
Status
Edit Settings
Edit Search Properties
Edit Search Object Types

Edit Search Properties

SaveReset

Name	Order	Description	Header(?)	Result(?)
alias	0	Alias	☒	☒
key	0	Component ID	☐	☐
name	0	Name	☐	☒

Reconfigure DocDB Properties

Reconfigure the search properties from the Peek-Admin site, under Plugins -> Document DB

Peek Admin Dashboard
Platform
Plugins
Product of Synerty Pty Ltd, All Rights Reserved
Version ###PEEKVER###

Home
Status
View Document
Edit Document Types
Edit Property Names
Edit Settings

Edit Search Properties

SaveReset

Model Set Key	Name	Order	Title	Header	Tooltip	Summary	Detail
pofDiagram	zone	0	Zone	☐	☐	☒	☒
pofDiagram	feeder cb source	0	Feeder CB Source	☐	☐	☐	☒
pofDiagram	alias	-3	Alias	☒	☒	☒	☒
pofDiagram	key	0	Component ID	☐	☐	☐	☒
pofDiagram	gis fid	0	GIS FID	☐	☐	☐	☒
pofDiagram	gxp source	0	GXP Source	☐	☐	☐	☒
pofDiagram	network asset id	0	Network Asset ID	☐	☐	☐	☒
pofDiagram	phases	0	Phases	☐	☐	☐	☒
pofDiagram	voltage	0	Voltage	☐	☐	☒	☒
pofDiagram	name	-2	Name	☐	☒	☐	☒
pofDiagram	current state	0	Current State	☐	☐	☐	☒

12.9.5 v2.0.5 Issues Log

Improvement

- [PEEK-450] - Diagram Edit - Hide overlays when in edit mode or when viewing branches.
- [PEEK-451] - Diagram Position - Provide config for default position on zoom level.
- [PEEK-452] - GraphDB - Upstream / Downstream is not implemented (It is now)
- [PEEK-453] - Diagram Tracer - Put a limit on the size of the trace

Bug

- [PEEK-448] - Diagram - Sometimes the diagram loads and is blank
- [PEEK-449] - Diagram - Performance fix for bounds updating

12.9.6 v2.0.4 Issues Log

Improvement

- [PEEK-444] - DocDB Generic Menu - Add generic menu support for condition !=
- [PEEK-436] - Core Search - Add order to Object Types so the result tabs can be ordered.
- [PEEK-439] - Diagram Edit - Support undo / redo

Bug

- [PEEK-434] - ENMAC Diagram Loader - float() argument must be a string or a number, not 'NoneType'
- [PEEK-435] - Core Search - `ujson.loads(objectPropsById[str(objectId)])` : Expected String or Unicode
- [PEEK-438] - DocDB Menu - Lookahead REXEXP doesn't work on safari
- [PEEK-440] - Diagram Trace - Only show trace if key is in graphdb
- [PEEK-447] - Diagram Edit - Saving a branch with a text disp that has no text throws an error

12.9.7 v2.0.3 Issues Log

Bug

- [PEEK-408] - GraphDB / Diagram - Trace complains about colours on slow networks (first load)
- [PEEK-409] - Desktop - If there are no config links, then don't show the config link button
- [PEEK-410] - Core Device - If no device id is entered, tell the user to enter one, instead of the exception.
- [PEEK-411] - VortexPY - If VM is suspended the vortex will timeout and force a restart of Peek
- [PEEK-412] - GIS Diagram - The diagram doesn't load.
- [PEEK-413] - DocDB - If there are no properties enabled then don't show the tooltip.
- [PEEK-414] - GIS/DMS Diagram - Switching between diagrams causes lots of errors and fails.
- [PEEK-415] - Core User - Update Hooks API so desktop logins don't trigger SOAP logins to PowerOn
- [PEEK-416] - Core User - "Failed to login : not enough arguments for format string" using Ldap

- [PEEK-417] - Core Search - “key” is not apart of the search object properties in the results
- [PEEK-418] - ENMAC Equipment Loader - District Zone field to import
- [PEEK-420] - DocDB Generic Menu - Add a condition to show the menu or not, use substitutions from the Doc
- [PEEK-421] - DocDB Generic Menu - Don’t show menu option if not all variables are filled.
- [PEEK-422] - Peek Server - Make recovery password show in config.json with out clicking login on site.
- [PEEK-423] - Peek Server - Memory issue on agent restart (suspected)
- [PEEK-424] - DocDB - Admin doc test screen seems not to work.
- [PEEK-425] - ENMAC Diagram Loader - Feeder colours does not work
- [PEEK-426] - Plugins have “ENMAC” in their names in the admin screen
- [PEEK-428] - “View Job” has not icon
- [PEEK-429] - Core Search - If keyword index fails, then it’s never retried
- [PEEK-430] - Core Search - results with no property display poorly
- [PEEK-431] - DocDB imports None/” values
- [PEEK-432] - ENMAC Diagram Loader - Deleting pages fails
- [PEEK-433] - Core User - Logic for not filling out an OU or a CN is buggy

Improvement

- [PEEK-427] - Create ENMAC Switching Job Loader

12.9.8 v2.0.1 Issues Log

Bug

- [PEEK-397] - Diagram Button Menu - missing some tooltips
- **[PEEK-399] - Print DMS Diagram - black sections of the canvas shown** in browser print preview
- [PEEK-400] - Markup Support View Branch - ANT Theme-ing TODO
- [PEEK-401] - Markup Support View Branch Items - browser unresponsive
- [PEEK-402] - DMS Diagram Markup Support - unable to edit existing branch
- [PEEK-405] - VortexJS - unable to begin transaction (3850 disk I/O error)
- **[PEEK-406] - Core User - Logging into the same browser with two browser** windows causes a logout

Improvement

- [PEEK-404] - DMS Diagram Markup Support - Check Save change before close
- [PEEK-233] - PERFORMANCE - SearchIndexChunkCompilerTask is slow

12.9.9 v2.0.0 Issues Log

Bug

- [PEEK-297] - Peek Desktop - Left Panel Appears unfinished
- [PEEK-298] - DMS Diagram - Remove DMS Diagram landing page
- [PEEK-299] - Pointer Cursor on Select World screen
- [PEEK-301] - Core Search / Diagram / ENMAC Diagram Loader - show on other world panel should use descriptions
- [PEEK-305] - Core Search - Hide panel after select show link
- [PEEK-306] - Core Search / Diagram / ENMAC Diagram Loader - hide show on link for current world
- [PEEK-308] - Core Search - Cleanup search results display
- [PEEK-309] - DocDB - Show Properties incomplete
- [PEEK-332] - Diagram Edit - Symbols need to rotate after insertion
- [PEEK-333] - Diagram Edit - Symbols to be selected as a whole
- [PEEK-334] - Peek to use HTTPS
- [PEEK-335] - Peek Server - Peek Admin Doesn't Require Authentication
- [PEEK-336] - Core-User - Restrict Users to a particular AD group
- [PEEK-347] - GraphDB - Running peek_logic_service causes massive memory leak.
- [PEEK-348] - Diagram - Add tooltips to view toolbar
- [PEEK-360] - GraphDB PowerOn Loader - unsupported 'datetime.datetime' and 'NoneType'
- [PEEK-361] - Diagram - Problem with Disp linked DispLayer not matching DispLayer in LookupService
- [PEEK-362] - Search - Property and Object Type fields are sometimes blank
- [PEEK-365] - Diagram Panel - Equipment Panel is just terrible, make it a popover
- [PEEK-367] - Peek fails to load in MS Edge
- [PEEK-393] - Diagram fails to position on, in Edge
- [PEEK-394] - IndexedDB is not open on Edge for diagram
- [PEEK-368] - Diagram Edit - Hide conductor template button in edit mode
- [PEEK-369] - Diagram Edit - Clicking on the items in the branch causes the browser to crash
- [PEEK-371] - Diagram Edit - When creating a new node, show a circle or something before the template is selected
- [PEEK-372] - Diagram - GridCache is not working.
- [PEEK-374] - Logged in on another device message
- [PEEK-379] - Diagram Generic Menu attributes not populating in URL
- [PEEK-381] - Diagram Panel - reduce the number of properties shown.
- [PEEK-383] - Diagram Panel - order the buttons shown by name
- [PEEK-385] - Diagram Panel - Too many properties shown in equipment info
- [PEEK-387] - All peek text is way to big in Peek Desktop

- [PEEK-395] - DocDB - New popups secondary menu falls below other modals (such as search)
- [PEEK-396] - Fix Angular errors preventing ng build –prod, and enable in Peek

Task

- [PEEK-341] - Add support for action delegates in proxy
- [PEEK-378] - Email NAR - disable send tab before saving

Improvement

- [PEEK-326] - Add support for partial keywords in search.
- [PEEK-351] - Implement websocket upgrades, so two ports are no longer required
- [PEEK-354] - Add in UI support for ant.design
- [PEEK-355] - Upgrade to Angular 8, etc
- [PEEK-366] - Core User - Add support for multiple browser logins
- [PEEK-389] - Upgrade docdb plugin properties, to reusable popups
- [PEEK-390] - Make “Show on diagram” item popup buttons dynamic again
- [PEEK-391] - Make DocDB popup screens configurable
- [PEEK-392] - Core User - Add alternate login form, suitable for desktops

12.10 v1.3.x Release Notes

12.10.1 Changes

Core Plugins

The following plugins were renamed and converted to core plugins:

- peek_plugin_search -> peek_core_search
- peek_plugin_user -> peek_core_user

These plugins were converted to provide better integration with the platform.

Diagram Generic Menu

This plugin has been renamed from “peek_plugin_generic_diagram_menu” to peek_plugin_diagram_generic_menu” for consistency with the new diagram trace plugin.

Diagram Zepben Menu

This plugin has been renamed from “peek_plugin_generic_zepben_menu” to peek_plugin_diagram_zepben_menu” for consistency with the new diagram trace plugin.

Branch Plugin

A new common branch plugin has been created. This plugin will be used to store the details of a branch. The plan is for other plugins that support branches in their models to use this plugin as a common reference, EG, enabling a branch in this plugin will enable it in the diagram, GraphDB, and DocDB

Diagram Branches

Diagram branches is a now fully implemented

The branches can be enabled or disabled, and are applied on top of the baseline diagram upon each render cycle.

Branches can be edited with the new Diagram Edit feature.

The diagram now also has a print view support and selectable layers.

The Diagram now requires a user login.

LDAP Authentication

The peek

RHEL7 Support

Peek now has installation instructions and support for Redhat Enterprise Linux 7 (RHEL7)

Required Dependency Bump

Peek required dependencies have been upgraded as follows:

- Oracle client 18.5
- openldap

OpenLDAP can be installed in MacOS with

```
brew install openldap
```

OR Debian Linux

```
apt-get install openldap-dev
```

or RHEL Linux

```
yum instal openldap-devel
```

Windows Services

Nil, carry on.

12.10.2 Linux Deployment

Nil

12.10.3 macOS Deployment

Nil

12.10.4 iOS Deployment

Peek v1.3.x does not have support for iOS, this will be updated in a future release

12.10.5 Windows Deployment

Nil.

Note: The windows deployment will change to use Windows Subsystem for Linux in a future release.

12.10.6 Enable New Plugins

Update the peek config.json files.

1. Edit each of C:\Users\peekpeek-XXXX\homeconfig.json
2. Add `peek_plugin_branch` to the start
3. Add `peek_plugin_enmac_email_nar` at the end

Start up the Peek Server service, it will rebuild the admin site.

Restart all Peek services.

For windows, restart the `peek-logic-service` service then start the `peek-restarter` service, the agent, worker and client will now start.

12.11 v1.2.x Release Notes

12.11.1 Changes

Graph DB

The new GraphDB plugin provides a connectivity model with trace configs and trace support.

The GraphDB has offline support allowing tracing to be run offline in the native mobile app.

ENMAC Connectivity Model Loader

The ENMAC Connectivity Model loader plugin extracts the connectivity model and the trace configuration from GEs PowerOn Fusion / PowerOn Advantage, the model and trace configs are loaded into the GraphDB.

The loader loads chunks of the connectivity model at a time, and requires “Split Points” to be configured.

Diagram Branches

Diagram branches is a new feature allowing plugins to modify the diagram displayed as it's being rendered.

Initially this includes changing the colours of shapes to provide trace highlighting support, but in the near future, this will be expanded to allow creating, deleting and moving shapes.

The branches can be enabled or disabled, and are applied on top of the baseline diagram upon each render cycle.

RHEL7 Support

Peek now has installation instructions and support for Redhat Enterprise Linux 7 (RHEL7)

Required Dependency Bump

Peek required dependencies have been upgraded as follows:

- Python 3.6.7

Optional Dependency Bump

Peeks optional dependencies have been upgraded as follows:

- Oracle client 12.1 -> Oracle client 18.3
- Python Package cx-Oracle==5.3 -> cx-Oracle>=7.0

Windows Services

Nil, carry on.

12.11.2 Linux Deployment

The Linux service scripts have been modified to use systemd, This is supported by both RHEL7 and Debian9. This change allows the scripts to work on both Linux distributions.

For upgrading from pre v1.2.x, you need to disable and remove the old init scripts on Debian.

```
for service in peek_logic_service peek_worker_service peek_agent_service peek_office_  
↪service  
do  
    service ${service} stop  
    update-rc.d ${service} disable  
    rm /etc/init.d/${service}  
done
```

12.11.3 macOS Deployment

Update to the latest XCode, 10.1.

12.11.4 iOS Deployment

The Peek Mobile native app now supports iOS 12.1.

12.11.5 Windows Deployment

nil.

12.11.6 Enable New Plugins

Update the peek config.json files.

1. Edit each of C:\Users\peekpeek-XXXX\homeconfig.json
2. Add `peek_plugin_graphdb` **after** `peek_plugin_livedb`
3. Add `peek_plugin_enmac_graphdb_loader` **after** `peek_plugin_enmac_diagram_loader`

Start up the Peek Server service, it will rebuild the admin site.

Connect to the admin site at <http://localhost:8010>

go to Plugins -> ENMAC Connectivity Model Loader

Select the “Edit App Server Settings” tab, enter the details and save.

Select the “Edit Graph Segments” tab, enter selection criteria for the connectivity model split pints, and save.

The agent needs to be restarted if it was already running.

Restart all Peek services.

For windows, restart the `peek-logic-service` service then start the `peek-restarter` service, the agent, worker and client will now start.

12.12 v1.1.0 Release Notes

12.12.1 Changes

Unified Search

Peek now has a new unified search plugin. This plugin is populated by other “loader” plugins with all kinds of information.

The search plugin handles the indexing and storage of all the information and provides a UI.

There is an Angular service API so other plugins can retrieve search results at will.

Search items consist of some key, value properties, and some paths to route to should the user select them.

The search plugin has full offline support, by default it’s online.

Document DB

Peek now has a new DocDB plugin, as in Json Document.

This plugin stores JSON documents and has a simple UI that presents key/values from the document.

The DocDB plugin handles all the storage, memory caching, compressing and transport to the clients.

There is an Angular service API so other plugins can retrieve documents at will.

The DocDB plugin has full offline support, by default it's online.

Offline Diagram

Not to be outdone by the search and docdb plugins, the diagram plugin now has full offline support as well for nativescript apps (web is supported, but it's disabled).

Like the other plugins, the diagram will download and store all diagram grids and lookups locally. It will check for updated grids every 15m and download the changes.

The "LocationIndex" in the diagram has had a small overhaul, previously it insisted on caching all the location index chunks to the browser/device before it would locate something. Now it supports online queries, significantly improving the speed of the initial diagram load.

The diagram now highlights equipment when it positions on them.

Finally, If you're using a web browser, the diagram updates the URL in the address bar, so you can share links or hit reload and the diagram will show restore to its previous state.

VortexJS

The VortexJS performance for the `TupleDataOfflineObserverService` class. This is the class that handles most of the locally/offline cached data that is reactively observed from the peek client.

There are performance and memory improvements, with the memory cached tuples now being purged after two minutes, and a significant reduction of the local storage save calls.

NativeScript UI

The nativescript UI responsiveness has been significantly improved.

Dependency Bump

Peek dependencies are upgrade as follows:

Python 3.6.6

Windows Services

Peek v1.1.0 now contains windows services. These release notes will describe how to install the services.

Windows Services

12.12.2 Linux Deployment

Nil, carry on being awesome.

12.12.3 macOS Deployment

Update to the latest XCode, 9.4.

12.12.4 Windows Deployment

This version of Peek upgrades several dependencies of the system. Follow these instructions to upgrade all the dependencies.

1. Uninstall Python
2. Delete the old Python install and peek virtual environments.
3. delete C:\Users\peek\Python36
4. delete C:\Users\peek\synerty-peek*
5. Reinstall the software again based on these instructions:
6. Install Python 3.6.6

Install PostgreSQL

Deploy the platform as per the synerty-peek instructions. Take note to answer Y and Y at the end to ensure the services are installed

Windows

12.12.5 Enable New Plugins

Update the peek config.json files.

1. Edit each of C:\Users\peek\peek-XXXX\homeconfig.json
 2. Add peek_core_docdb after peek_plugin_livedb
 3. Add peek_core_search after peek_plugin_livedb
 4. Add peek_plugin_enmac_equipment_loader after peek_plugin_enmac_diagram_loader
-

Start up the Peek Server service, it will rebuild the admin site.

Connect to the admin site at <http://localhost:8010>

go to Plugins -> ENMAC Equipment Detail Loader

Select the “Edit App Server Settings” tab, enter the details and save.

The agent needs to be restarted if it was already running.

Restart all Peek services.

For windows, restart the `peek-logic-service` service then start the `peek-restarter` service, the agent, worker and client will now start.

12.13 v0.10.0 Release Notes

12.13.1 Changes

Vortex PayloadEnvelope

This version of peek contains some breaking changes to do with the VortexJS/VortexPY.

A new class called “PayloadEnvelope” has been introduced. PayloadEnvelope wraps a Payload and is routed around the Vortexes.

The `toVortexMsg/fromVortexMsg` methods on the Payload class have been renamed to `toEncodedPayload/fromEncodedPayload` respectively.

This change was made to improve performance, in some instances the `Payload.tuples` didn’t need to be deserialised/reserialised, for example when passing through the `peek_office_service` service, or being cached in the browsers/mobile devices.

Dependency Bump

Peek dependencies are upgrade as follows:

1. Python 3.6.5
2. PostgreSQL 10.4
3. MsysGit - Install settings change

Windows Services

Peek v0.10.0 now contains windows services. These release notes will describe how to install the services.

12.13.2 Deployment

This version of Peek upgrades several dependencies of the system. Follow these instructions to upgrade all the dependencies.

First, backup the PostgreSQL peek database.

Delete the virtual environments

delete C:\Users\peek\synerty-peek*

1. Uninstall Python
2. Uninstall “Git version”

3. Uninstall PostgreSQL

Delete the old PostgreSQL database data directory.

delete C:\Program Files\PostgreSQL

delete C:\Users\peek\Python36

Reinstall the software again based on these instructions:

1. Install Python
2. Install Msys Git
3. Install PostgreSQL

Setup OS Requirements Windows

Open PGAdmin4, and restore the database backup.

Note: Ensure you restore the database to the `peek` database (not the postgres one)

Deploy the platform, Y and Y at the end.

Deploy Peek Release

These steps grant “Login as Service” to the “.peek” user

1. Run “services.msc”
2. Find the peek server service
3. Open the properties of the service
4. Goto the LogOn tab
5. Enter the password twice and hit OK
6. A dialog box will appear saying that the Peek users has been granted the right.

12.14 v0.6.0 Release Notes

The following modifications are required to upgrade plugins to run on the v0.6.0 version of the platform

12.14.1 NPM : peek-field-app-util

The `peek-field-app-util` npm packages has been renamed to `peek-util`.

Run the following to help with the upgrade


```
find ./ -name "*.ts" -not -name "node_modules" -exec sed -i 's/peek-field-app-util/  
↳peek-util/g' {} \;
```

The peek-field-app-util/index.nativescript typescript index file has been renamed to peek-util/index.ns

Run the following to help with the upgrade

```
find ./ -name "*.ts" -not -name "node_modules" -exec sed -i 's,peek-util/index.  
↳nativescript,peek-util/index.ns,g' {} \;
```


CHAPTER 13

Indices and tables

- `genindex`
- `modindex`
- `search`

p

peek_plugin_base, 319
peek_plugin_base.agent, 319
peek_plugin_base.agent.PeekAgentPlatformHookABC, 319
peek_plugin_base.agent.PeekPlatformAgentHttpHookABC, 319
peek_plugin_base.agent.PluginAgentEntryHookABC, 320
peek_plugin_base.client, 320
peek_plugin_base.client.PeekClientPlatformHookABC, 320
peek_plugin_base.client.PeekPlatformFieldHttpHookABC, 320
peek_plugin_base.client.PeekPlatformOfficeHttpHookABC, 321
peek_plugin_base.client.PluginClientEntryHookABC, 322
peek_plugin_base.PeekPlatformCommonHookABC, 331
peek_plugin_base.PeekPlatformFileStorageHookABC, 331
peek_plugin_base.PeekPlatformServerInfoHookABC, 332
peek_plugin_base.PeekVortexUtil, 332
peek_plugin_base.PluginCommonEntryHookABC, 333
peek_plugin_base.PluginPackageFileConfig, 333
peek_plugin_base.server, 322
peek_plugin_base.server.PeekPlatformAdminHttpHookABC, 322
peek_plugin_base.server.PeekPlatformServerHttpHookABC, 323
peek_plugin_base.server.PeekServerPlatformHookABC, 324
peek_plugin_base.server.PluginLogicEntryHookABC, 324
peek_plugin_base.server.PluginServerStorageEntryHookABC, 324
peek_plugin_base.server.PluginServerWorkerEntryHookABC, 325
peek_plugin_base.storage, 325
peek_plugin_base.storage.AlembicEnvBase, 325
peek_plugin_base.storage.DbConnection, 326
peek_plugin_base.storage.LoadPayloadPgUtil, 327
peek_plugin_base.storage.RunPyInPg, 328
peek_plugin_base.storage.StorageUtil, 328
peek_plugin_base.storage.TypeDecorators, 329
peek_plugin_base.worker, 329
peek_plugin_base.worker.CeleryApp, 329
peek_plugin_base.worker.CeleryDbConn, 330
peek_plugin_base.worker.CeleryDbConnInit, 330
peek_plugin_base.worker.PeekWorkerPlatformHookABC, 330
peek_plugin_base.worker.PluginWorkerEntryHookABC, 331

A

addAdminResource()
 (peek_plugin_base.server.PeekPlatformAdminHttpHookABC.
 method), 322
 addAdminStaticResourceDir()
 (peek_plugin_base.server.PeekPlatformAdminHttpHookABC.
 method), 323
 addAgentExternalApiResource()
 (peek_plugin_base.agent.PeekPlatformAgentHttpHookABC.
 method), 319
 addFieldResource()
 (peek_plugin_base.client.PeekPlatformFieldHttpHookABC.
 method), 321
 addFieldStaticResourceDir()
 (peek_plugin_base.client.PeekPlatformFieldHttpHookABC.
 method), 321
 addOfficeResource()
 (peek_plugin_base.client.PeekPlatformOfficeHttpHookABC.
 method), 321
 addOfficeStaticResourceDir()
 (peek_plugin_base.client.PeekPlatformOfficeHttpHookABC.
 method), 321
 addServerResource()
 (peek_plugin_base.server.PeekPlatformServerHttpHookABC.
 method), 323
 addServerStaticResourceDir()
 (peek_plugin_base.server.PeekPlatformServerHttpHookABC.
 method), 323
 AlembicEnvBase (class in
 peek_plugin_base.storage.AlembicEnvBase),
 325
 angularFrontendAppDir
 (peek_plugin_base.client.PluginClientEntryHookABC.
 attribute), 322
 angularMainModule
 (peek_plugin_base.client.PluginClientEntryHookABC.
 attribute), 322

B

bind_expression()
 (peek_plugin_base.server.PeekPlatformAdminHttpHookABC.
 method), 329
 CeleryCorators.PeekLargeBinary
 method), 329
 celeryAppIncludes
 (peek_plugin_base.worker.PluginWorkerEntryHookABC.
 method), 331
 checkForeignKeys()
 (peek_plugin_base.storage.DbConnection.DbConnection
 method), 327
 closeAllSessions()
 (peek_plugin_base.storage.DbConnection.DbConnection
 method), 327
 config(peek_plugin_base.PluginPackageFileConfig.PluginPackageFileC
 attribute), 333
 convert_to_python_module
 (peek_plugin_base.storage.DbConnection),
 327
 connect_to_db(peek_plugin_base.storage.LoadPayloadPgUtil.LoadPayloadTupleL
 attribute), 327
 DbConnection (class in
 peek_plugin_base.storage.DbConnection),
 326
 dbConnectString(peek_plugin_base.server.PeekServerPlatformHookA
 attribute), 324
 dbEngine(peek_plugin_base.server.PluginServerStorageEntryHookABC.
 attribute), 324
 dbEngine(peek_plugin_base.storage.DbConnection.DbConnection
 attribute), 326
 dbMetadata(peek_plugin_base.server.PluginServerStorageEntryHookAL
 attribute), 324
 dbSession(peek_plugin_base.server.PluginLogicEntryHookABC.Plugin
 attribute), 324
 dbSessionFactory(peek_plugin_base.server.PluginServerStorageEntr
 attribute), 325

E

encodedPayload(*peek_plugin_base.storage.LoadPayloadPgUtil*,
attribute), 327
ensureSchemaExists() (in module
peek_plugin_base.storage.AlembicEnvBase),
325

F

fileStorageDirectory
(*peek_plugin_base.PeekPlatformFileStorageHookABC*.*PeekPlatformFileStorageHookABC*
attribute), 331

G

getDbEngine() (in module
peek_plugin_base.worker.CeleryDbConn),
330
getDbSession() (in module
peek_plugin_base.worker.CeleryDbConn),
330
getOtherPluginApi()
(*peek_plugin_base.PeekPlatformCommonHookABC*.*PeekPlatformCommonHookABC*
method), 331
getTuplesPayload() (in module
peek_plugin_base.storage.LoadPayloadPgUtil),
327
getTuplesPayloadBlocking() (in module
peek_plugin_base.storage.LoadPayloadPgUtil),
327

I

impl(*peek_plugin_base.storage.TypeDecorators.PeekLargeBinary*,
attribute), 329
initWorkerConnString() (in module
peek_plugin_base.worker.CeleryDbConnInit),
330
initWorkerProcessDbConn() (in module
peek_plugin_base.worker.CeleryDbConnInit),
330
isMssqlDialect() (in module
peek_plugin_base.storage.AlembicEnvBase),
325
isPostGreSQLDialect() (in module
peek_plugin_base.storage.AlembicEnvBase),
325

L

load() (*peek_plugin_base.PluginCommonEntryHookABC*.*PluginCommonEntryHookABC*
method), 333
LoadPayloadTupleResult (class in
peek_plugin_base.storage.LoadPayloadPgUtil),
327

M

makeCoreValuesSubqueryCondition() (in

module peek_plugin_base.storage.StorageUtil),
328
makeOrmValuesSubqueryCondition() (in mod-
ule *peek_plugin_base.storage.StorageUtil*), 328
migrate() (*peek_plugin_base.storage.DbConnection.DbConnection*
method), 326
migrateStorageSchema()
(*peek_plugin_base.server.PluginLogicEntryHookABC*.*PluginLogicEntryHookABC*
method), 324

N

name(*peek_plugin_base.PluginCommonEntryHookABC*.*PluginCommonEntryHookABC*
attribute), 333

O

ormSessionCreator
(*peek_plugin_base.storage.DbConnection.DbConnection*
attribute), 326

P

peek_plugin_base(*module*), 319
peek_plugin_base.agent(*module*), 319
peek_plugin_base.agent.PeekAgentPlatformHookABC
(*module*), 319
peek_plugin_base.agent.PeekPlatformAgentHttpHookABC
(*module*), 319
peek_plugin_base.agent.PluginAgentEntryHookABC
(*module*), 320
peek_plugin_base.client(*module*), 320
peek_plugin_base.client.PeekClientPlatformHookABC
(*module*), 320
peek_plugin_base.client.PeekPlatformFieldHttpHookABC
(*module*), 320
peek_plugin_base.client.PeekPlatformOfficeHttpHookABC
(*module*), 321
peek_plugin_base.client.PluginClientEntryHookABC
(*module*), 322
peek_plugin_base.PeekPlatformCommonHookABC
(*module*), 331
peek_plugin_base.PeekPlatformFileStorageHookABC
(*module*), 331
peek_plugin_base.PeekPlatformServerInfoHookABC
(*module*), 332
peek_plugin_base.PeekVortexUtil (*module*),
332
peek_plugin_base.PluginCommonEntryHookABC
(*module*), 333
peek_plugin_base.PluginPackageFileConfig
(*module*), 333
peek_plugin_base.server(*module*), 322
peek_plugin_base.server.PeekPlatformAdminHttpHookABC
(*module*), 322

peek_plugin_base.server.PeekPlatformServerHttpHookABC (class in peek_plugin_base.server.PeekPlatformAdminHttpHookABC), (module), 323

peek_plugin_base.server.PeekServerPlatformHookABC (class in peek_plugin_base.agent.PeekPlatformAgentHttpHookABC), (module), 324

peek_plugin_base.server.PluginLogicEntryHookABC (class in PeekPlatformCommonHookABC (class in peek_plugin_base.PeekPlatformCommonHookABC), (module), 324

peek_plugin_base.server.PluginServerStorageEntryHookABC (class in peek_plugin_base.PeekPlatformCommonHookABC), (module), 324

peek_plugin_base.server.PluginServerWorkerEntryHookABC (class in peek_plugin_base.client.PeekPlatformFieldHttpHookABC), (module), 325

peek_plugin_base.storage (module), 325

peek_plugin_base.storage.AlembicEnvBase (class in peek_plugin_base.PeekPlatformFileStorageHookABC), (module), 325

peek_plugin_base.storage.DbConnection (class in PeekPlatformOfficeHttpHookABC (class in peek_plugin_base.client.PeekPlatformOfficeHttpHookABC), (module), 326

peek_plugin_base.storage.LoadPayloadPgUtil (class in PeekPlatformServerHttpHookABC (class in peek_plugin_base.server.PeekPlatformServerHttpHookABC), (module), 327

peek_plugin_base.storage.RunPyInPg (module), 328

peek_plugin_base.storage.StorageUtil (class in PeekPlatformServerInfoHookABC (class in peek_plugin_base.PeekPlatformServerInfoHookABC), (module), 328

peek_plugin_base.storage.TypeDecorators (class in peek_plugin_base.PeekPlatformServerInfoHookABC), (module), 329

peek_plugin_base.worker (module), 329

peek_plugin_base.worker.CeleryApp (module), 329

peek_plugin_base.worker.CeleryDbConn (class in peek_plugin_base.PeekPlatformServerInfoHookABC.PeekPlatformServerInfoHookABC), (module), 330

peek_plugin_base.worker.CeleryDbConnInit (class in peek_plugin_base.PeekPlatformServerInfoHookABC.PeekPlatformServerInfoHookABC), (module), 330

peek_plugin_base.worker.PeekWorkerPlatformHookABC (class in peek_plugin_base.PeekVortexUtil), 332

peek_plugin_base.worker.PluginWorkerEntryHookABC (class in peek_plugin_base.server.PeekServerPlatformHookABC), (module), 331

peekAdminName (in module peek_plugin_base.PeekVortexUtil), 332

peekAgentName (in module peek_plugin_base.PeekVortexUtil), 332

PeekAgentPlatformHookABC (class in peek_plugin_base.agent.PeekAgentPlatformHookABC), 319

PeekClientPlatformHookABC (class in peek_plugin_base.client.PeekClientPlatformHookABC), 320

peekDesktopName (in module peek_plugin_base.PeekVortexUtil), 332

peekFieldName (in module peek_plugin_base.PeekVortexUtil), 332

PeekLargeBinary (class in peek_plugin_base.storage.TypeDecorators), 329

peekMobileName (in module peek_plugin_base.PeekVortexUtil), 332

PeekPlatformAdminHttpHookABC (class in peek_plugin_base.server.PeekPlatformAdminHttpHookABC), 322

PeekPlatformAgentHttpHookABC (class in peek_plugin_base.agent.PeekPlatformAgentHttpHookABC), 324

PeekPlatformCommonHookABC (class in peek_plugin_base.PeekPlatformCommonHookABC), 331

PeekPlatformFieldHttpHookABC (class in peek_plugin_base.client.PeekPlatformFieldHttpHookABC), 325

PeekPlatformFileStorageHookABC (class in peek_plugin_base.PeekPlatformFileStorageHookABC), 331

PeekPlatformOfficeHttpHookABC (class in peek_plugin_base.client.PeekPlatformOfficeHttpHookABC), 321

PeekPlatformServerHttpHookABC (class in peek_plugin_base.server.PeekPlatformServerHttpHookABC), 323

PeekPlatformServerInfoHookABC (class in peek_plugin_base.PeekPlatformServerInfoHookABC), 332

peekServerHost (peek_plugin_base.PeekPlatformServerInfoHookABC attribute), 332

peekServerHttpPort (peek_plugin_base.PeekPlatformServerInfoHookABC.PeekPlatformServerInfoHookABC attribute), 332

peekStorageName (in module peek_plugin_base.PeekVortexUtil), 332

peekWorkerName (in module peek_plugin_base.PeekVortexUtil), 332

PeekWorkerPlatformHookABC (class in peek_plugin_base.worker.PeekWorkerPlatformHookABC), 330

pgCopyInsert () (in module peek_plugin_base.storage.DbConnection), 327

platform (peek_plugin_base.agent.PluginAgentEntryHookABC.PluginAgentEntryHookABC attribute), 320

platform (peek_plugin_base.client.PluginClientEntryHookABC.PluginClientEntryHookABC attribute), 322

platform (peek_plugin_base.server.PluginLogicEntryHookABC.PluginLogicEntryHookABC attribute), 324

platform (peek_plugin_base.worker.PluginWorkerEntryHookABC.PluginWorkerEntryHookABC attribute), 331

PluginAgentEntryHookABC (class in peek_plugin_base.agent.PluginAgentEntryHookABC), 320

PluginClientEntryHookABC (class in attribute), 322
 peek_plugin_base.client.PluginClientEntryHookABC, 322
 PluginCommonEntryHookABC (class in attribute), 323
 peek_plugin_base.PluginCommonEntryHookABC, 333
 run () (peek_plugin_base.storage.AlembicEnvBase.AlembicEnvBase method), 325
 PluginLogicEntryHookABC (class in runPyInPg () (in module
 peek_plugin_base.server.PluginLogicEntryHookABC), peek_plugin_base.storage.RunPyInPg), 328
 324
 runPyInPgBlocking () (in module
 PluginPackageFileConfig (class in peek_plugin_base.storage.RunPyInPg), 328
 peek_plugin_base.PluginPackageFileConfig), 333

S

PluginServerStorageEntryHookABC (class in serviceId(peek_plugin_base.PeekPlatformCommonHookABC.PeekPlatformCommonHookABC), 331
 peek_plugin_base.server.PluginServerStorageEntryHookABC, 324
 setConnStringForWindows () (in module
 PluginServerWorkerEntryHookABC (class in peek_plugin_base.worker.CeleryDbConn), 330
 peek_plugin_base.server.PluginServerWorkerEntryHookABC, 325
 shutdownWorkerProcessDbConn () (in module
 PluginWorkerEntryHookABC (class in peek_plugin_base.worker.CeleryDbConnInit), 330
 peek_plugin_base.worker.PluginWorkerEntryHookABC), 331
 prefetchDeclarativeIds () (in module start () (peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC), 333
 peek_plugin_base.worker.CeleryDbConn), stop () (peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC), 333
 330
 prefetchDeclarativeIds ()
 (peek_plugin_base.server.PluginServerStorageEntryHookABC.PluginServerStorageEntryHookABC method), 325
 taskEndCloseSession () (in module
 prefetchDeclarativeIds () (peek_plugin_base.storage.DbConnection.DbConnection peek_plugin_base.worker.CeleryDbConnInit), 330
 method), 326
 title(peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC), 333
 publishedAgentApi (peek_plugin_base.agent.PluginAgentEntryHookABC.PluginAgentEntryHookABC attribute), 320

U

publishedClientApi (peek_plugin_base.client.PluginClientEntryHookABC.PluginClientEntryHookABC attribute), 322
 unload () (peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC method), 333
 publishedServerApi (peek_plugin_base.server.PluginLogicEntryHookABC.PluginLogicEntryHookABC attribute), 324

R

rootAdminResource (peek_plugin_base.server.PeekPlatformAdminHttpHookABC.PeekPlatformAdminHttpHookABC attribute), 323
 rootAgentResource (peek_plugin_base.agent.PeekPlatformAgentHttpHookABC.PeekPlatformAgentHttpHookABC attribute), 320
 rootDir (peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC attribute), 333
 rootFieldResource (peek_plugin_base.client.PeekPlatformFieldHttpHookABC.PeekPlatformFieldHttpHookABC attribute), 321
 rootOfficeResource (peek_plugin_base.client.PeekPlatformOfficeHttpHookABC.PeekPlatformOfficeHttpHookABC attribute), 321