Synerty Peek Documentation

Release 2.0.0

Synerty

Aug 01, 2019

Contents:

1 How to Use Peek Documentation **1**

2 Overview **3**
- 2.1 Architecture . . . 3
- 2.2 Services . . . 4
  - 2.2.1 Server Service . . . 5
  - 2.2.2 Storage Service . . . 5
  - 2.2.3 Client Service . . . 5
  - 2.2.4 Mobile Service . . . 6
  - 2.2.5 Desktop Service . . . 7
  - 2.2.6 Worker Service . . . 7
  - 2.2.7 Agent Service . . . 7
  - 2.2.8 Admin Service . . . 8
- 2.3 Plugins . . . 8
  - 2.3.1 Enterprise Extensible . . . 8
  - 2.3.2 One Plugin, One Package . . . 9
- 2.4 Noop Plugin Example . . . 12

3 Setup OS Requirements **15**
- 3.1 Setup OS Requirements Windows . . . 15
  - 3.1.1 Installation Objective . . . 15
  - 3.1.2 OS Commands . . . 16
  - 3.1.3 Installation Guide . . . 16
  - 3.1.4 Create Peek OS User . . . 16
  - 3.1.5 MS .NET Framework 3.5 SP1 . . . 17
  - 3.1.6 Visual C++ Build Tools 2015 . . . 17
  - 3.1.7 Setup Msys Git . . . 17
  - 3.1.8 Install PostgresSQL . . . 18
  - 3.1.9 Install Python 3.6 . . . 22
  - 3.1.10 Install Worker Dependencies . . . 24
  - 3.1.11 Install Oracle Client (Optional) . . . 25
  - 3.1.12 Install FreeTDS (Optional) . . . 26
  - 3.1.13 What Next? . . . 27
- 3.2 Setup OS Requirements MacOS . . . 27
  - 3.2.1 Installation Objective . . . 27
  - 3.2.2 Installation Guide . . . 28
  - 3.2.3 Safari Open Safe Files . . . 28

3.2.4 Create Peek Platform OS User . . . 29
3.2.5 Install Xcode . . . 30
3.2.6 Install an Oracle JDK . . . 30
3.2.7 Install Homebrew . . . 30
3.2.8 Install PostGreSQL . . . 31
3.2.9 Install Python 3.6 . . . 32
3.2.10 Install Worker Dependencies . . . 34
3.2.11 Install Oracle Client (Optional) . . . 35
3.2.12 Install FreeTDS (Optional) . . . 36
3.2.13 Change Open File Limit on macOS . . . 37
3.2.14 What Next? . . . 37
3.3 Setup OS Requirements RHEL . . . 37
3.3.1 Installation Objective . . . 37
3.3.2 Installation Guide . . . 38
3.3.3 Install Red Hat Linux Server 7.6 OS . . . 38
3.3.4 Login as Peek . . . 54
3.3.5 Registering RHEL . . . 54
3.3.6 Removing IPv6 Localhost . . . 54
3.3.7 Installing General Prerequisites . . . 55
3.3.8 Installing VMWare Tools (Optional) . . . 56
3.3.9 Update Firewall . . . 57
3.3.10 Install PostGreSQL . . . 57
3.3.11 Compile and Install Python 3.6 . . . 58
3.3.12 Install Worker Dependencies . . . 60
3.3.13 Install Oracle Client (Optional) . . . 62
3.3.14 Install FreeTDS (Optional) . . . 62
3.3.15 What Next? . . . 63
3.4 Setup OS Requirements Debian . . . 63
3.4.1 Installation Objective . . . 64
3.4.2 Installation Guide . . . 64
3.4.3 Install Debian 8 OS . . . 64
3.4.4 SSH Setup . . . 69
3.4.5 Login as Peek . . . 70
3.4.6 Configure Static IP (Optional) . . . 70
3.4.7 Installing General Prerequisites . . . 70
3.4.8 Installing VMWare Tools (Optional) . . . 72
3.4.9 Install PostGreSQL . . . 72
3.4.10 Compile and Install Python 3.6 . . . 74
3.4.11 Install Worker Dependencies . . . 75
3.4.12 Install Oracle Client (Optional) . . . 76
3.4.13 Install FreeTDS (Optional) . . . 77
3.4.14 What Next? . . . 78
3.5 What Next? . . . 78

4 Deploy Peek Platform **79**
4.1 Windows . . . 79
4.1.1 Deploy Virtual Environment . . . 79
4.2 Linux . . . 81
4.3 macOS . . . 81
4.4 Development Considerations . . . 82
4.5 What Next? . . . 82

5 Deploy Peek Plugins **83**
5.1 Deploying a Production Release . . . 83

5.2 What Next? . . . 84

6 Administration **85**
6.1 Configuring Platform `config.json` . . . 85
6.1.1 Peek Server . . . 85
6.1.2 Peek Client . . . 86
6.1.3 Peek Agent . . . 87
6.2 Run Peek Manually . . . 87
6.2.1 Check Environment . . . 87
6.2.2 run_peek_server . . . 88
6.2.3 run_peek_client . . . 88
6.2.4 run_peek_agent . . . 88
6.2.5 Whats Next . . . 88
6.3 Logs . . . 89
6.4 Getting Support . . . 89
6.4.1 Enterprise Support . . . 89
6.4.2 Reporting Bugs . . . 89
6.5 Updating Plugin Settings . . . 90
6.6 Peek Windows Admin . . . 90
6.6.1 Windows Services . . . 90
6.6.2 Backup and Restore PostGreSQL DB . . . 90

7 NativeScript App **93**
7.1 Setup Nativescript Windows . . . 93
7.1.1 Installation Objective . . . 93
7.1.2 Online Installation Guide . . . 94
7.1.3 What Next? . . . 96
7.2 Setup Nativescript Debian . . . 96
7.2.1 Installation Objective . . . 96
7.2.2 Installation Guide . . . 97
7.2.3 What Next? . . . 99
7.3 Setup Nativescript MacOS . . . 99
7.3.1 Installation Objective . . . 99
7.3.2 Installation Guide . . . 100
7.3.3 Nativescript Package . . . 101
7.3.4 Android Emulator Setup . . . 103
7.3.5 Android Emulator Setup for VM . . . 105
7.3.6 What Next? . . . 105
7.4 Package NativeScript App . . . 105
7.4.1 Windows . . . 106
7.4.2 Linux . . . 106
7.4.3 What Next? . . . 106
7.5 Deploy NativeScript App . . . 106
7.5.1 Windows . . . 107
7.5.2 Linux . . . 107
7.5.3 What Next? . . . 107
7.6 What Next? . . . 107

8 Peek Development **109**
8.1 Develop Peek Plugin Guides . . . 109
8.1.1 Develop Peek Plugins . . . 109
8.1.2 Setup Plugin for Development . . . 114
8.1.3 Developing With The Frontends . . . 115
8.1.4 Continue Development . . . 121

8.1.5 What Next? . . . 121
8.2 Publish Peek Plugins . . . 121
8.2.1 Create Private Plugin Release . . . 121
8.2.2 Create PyPI Public Release . . . 122
8.2.3 What Next? . . . 123
8.3 Package Peek Plugins . . . 123
8.3.1 Packaging a Production Release . . . 123
8.3.2 What Next? . . . 125
8.4 Develop Peek Platform . . . 125
8.4.1 Development Setup Objective . . . 126
8.4.2 Hardware Recommendation . . . 126
8.4.3 Software Installation and Configuration . . . 126
8.4.4 What Next? . . . 129
8.5 Publish Peek Platform . . . 129
8.5.1 Building a Production Release . . . 129
8.5.2 Building a Development Release . . . 129
8.5.3 What Next? . . . 130
8.6 Package Peek Platform . . . 130
8.6.1 Building a Windows Release . . . 130
8.6.2 Building a Linux Release . . . 131
8.6.3 Building a macOS Release . . . 131
8.6.4 What Next? . . . 132
8.7 Setup Pycharm IDE . . . 132
8.8 Setup VS Code IDE . . . 136
8.9 Learn Plugin Development . . . 136
8.9.1 First Steps . . . 136
8.9.2 Scaffolding From Scratch . . . 138
8.9.3 Add Documentation . . . 145
8.9.4 Add Server Service . . . 161
8.9.5 Add Client Service . . . 165
8.9.6 Add Agent Service . . . 168
8.9.7 Add Storage Service . . . 171
8.9.8 Add Admin Service . . . 180
8.9.9 Add Mobile Service . . . 184
8.9.10 Add Tuples . . . 196
8.9.11 Add Tuple Loader . . . 202
8.9.12 Add Offline Storage . . . 214
8.9.13 Add Observables . . . 216
8.9.14 Add Actions . . . 229
8.9.15 Add Vortex RPC . . . 244
8.9.16 Add Plugin Python API . . . 255
8.9.17 Add Plugin TypeScript APIs (TODO) . . . 261
8.9.18 Use Plugin APIs (TODO) . . . 261
8.9.19 Add Worker Service . . . 261
8.10 What Next? . . . 264

9 Troubleshooting 265
9.1 Troubleshooting Windows . . . 265
9.1.1 Test cx_Oracle in Python . . . 265
9.2 Troubleshooting Debian . . . 265
9.2.1 Test cx_Oracle in Python . . . 265
9.2.2 OSError: inotify instance limit reached . . . 266
9.3 Troubleshooting macOS . . . 266
9.3.1 Test cx_Oracle in Python . . . 266

9.3.2 ORA-21561: OID generation failed . . . 266

10 Utilities **269**

11 API Reference **271**
11.1 File `plugin_package.json` . . . 271
11.2 Synerty Peek . . . 271
11.2.1 peek_plugin_base . . . 271

12 Upgrade Notes **285**
12.1 v2.0.x Upgrade Notes . . . 285
12.1.1 Changes . . . 285
12.1.2 Linux Deployment . . . 287
12.1.3 macOS Deployment . . . 287
12.1.4 iOS Deployment . . . 287
12.1.5 Windows Deployment . . . 287
12.1.6 Enable New Plugins . . . 287
12.1.7 Issues Log . . . 288
12.2 v1.3.x Upgrade Notes . . . 288
12.2.1 Changes . . . 288
12.2.2 Linux Deployment . . . 290
12.2.3 macOS Deployment . . . 290
12.2.4 iOS Deployment . . . 290
12.2.5 Windows Deployment . . . 290
12.2.6 Enable New Plugins . . . 290
12.3 v1.2.x Upgrade Notes . . . 291
12.3.1 Changes . . . 291
12.3.2 Linux Deployment . . . 292
12.3.3 macOS Deployment . . . 292
12.3.4 iOS Deployment . . . 292
12.3.5 Windows Deployment . . . 292
12.3.6 Enable New Plugins . . . 292
12.4 v1.1.0 Upgrade Notes . . . 293
12.4.1 Changes . . . 293
12.4.2 Linux Deployment . . . 294
12.4.3 macOS Deployment . . . 294
12.4.4 Windows Deployment . . . 294
12.4.5 Enable New Plugins . . . 295
12.5 v0.10.0 Upgrade Notes . . . 295
12.5.1 Changes . . . 295
12.5.2 Deployment . . . 296
12.6 v0.6.0 Upgrade Notes . . . 297
12.6.1 NPM : peek-mobile-util . . . 297

13 Indices and tables **299**

Python Module Index **301**

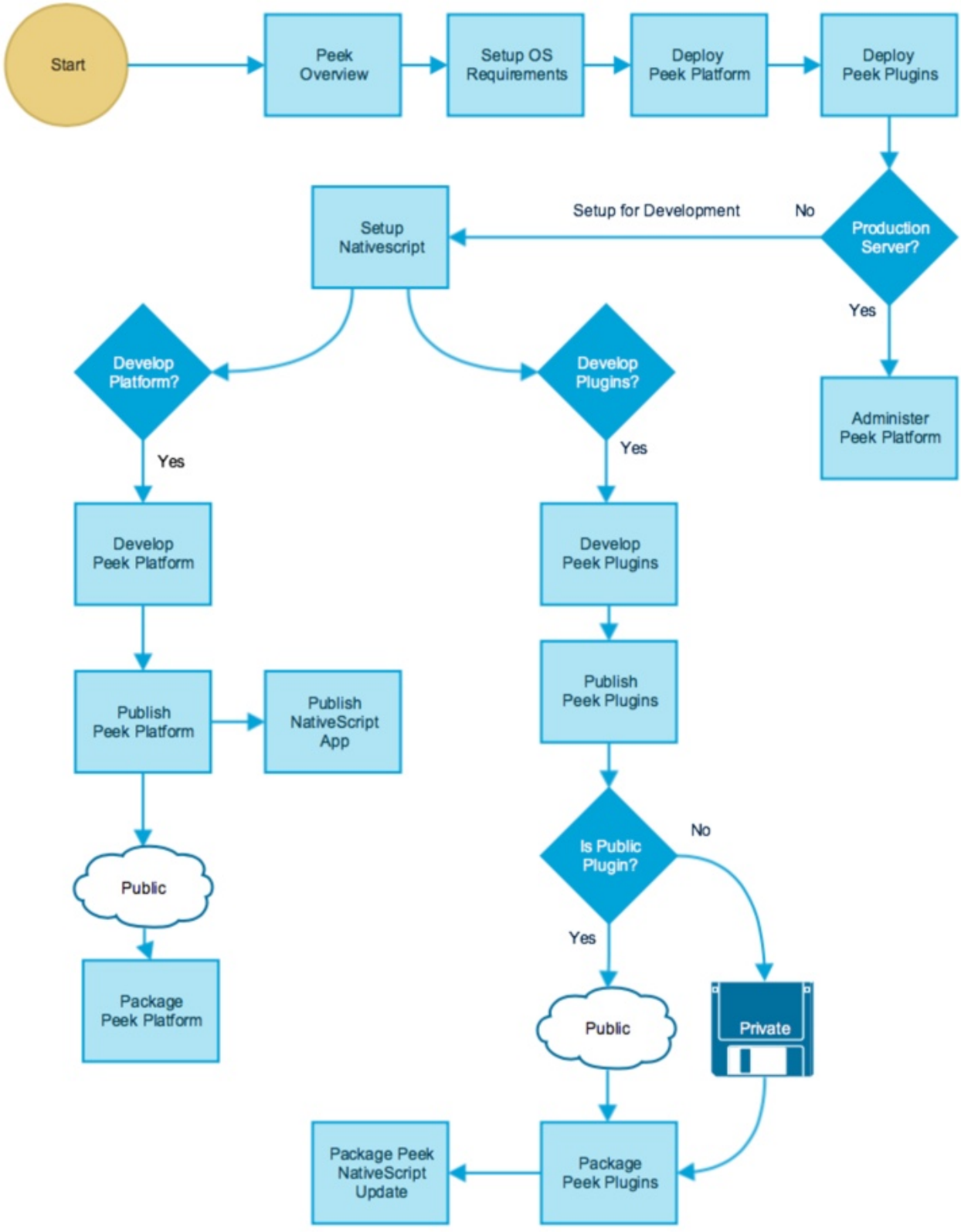
Index **303**

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.


Start
Peek Overview
Setup OS Requirements
Deploy Peek Platform
Deploy Peek Plugins
Production Server?
No
Setup for Development
Setup Nativescript
Yes
Administer Peek Platform
Develop Platform?
Develop Plugins?
Yes
Yes
Develop Peek Platform
Develop Peek Plugins
Publish Peek Platform
Publish NativeScript App
Publish Peek Plugins
Is Public Plugin?
No
Public
Yes
Package Peek Platform
Public
Private
Package Peek NativeScript Update
Package Peek Plugins

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 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.

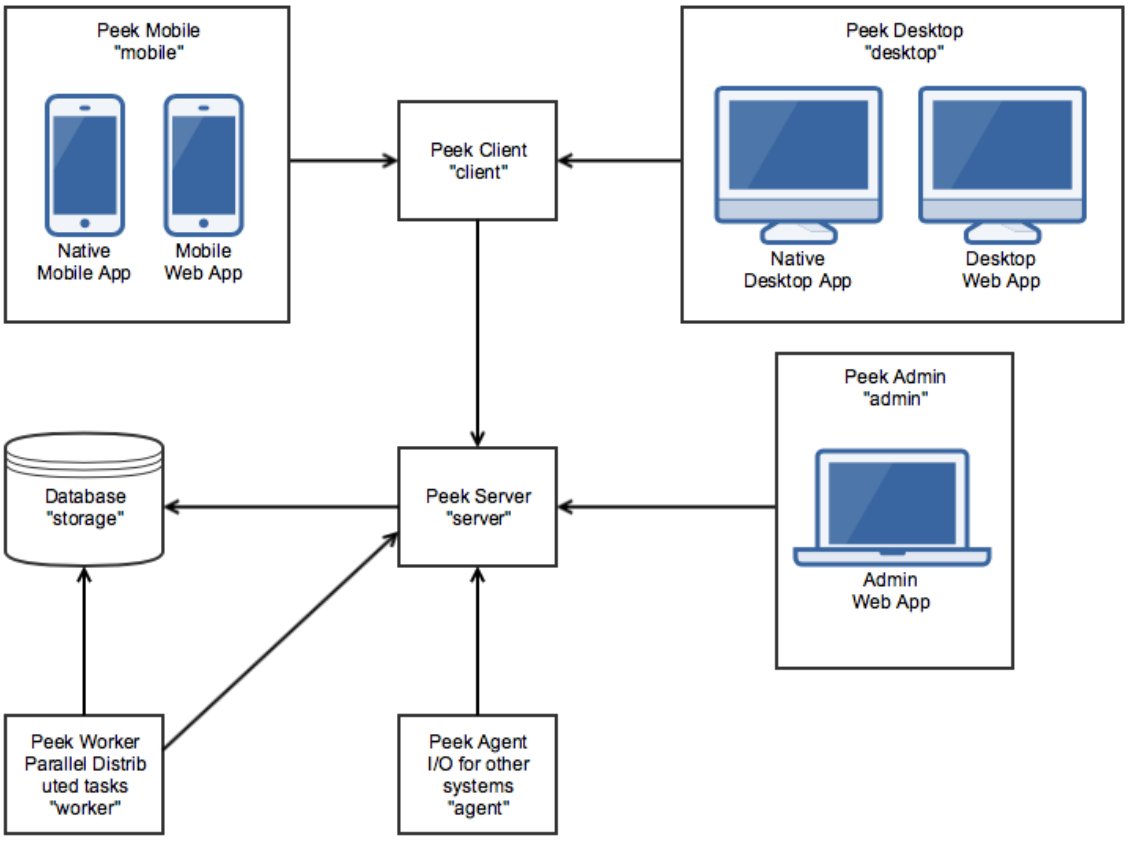
2.1 Architecture

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 Server.

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 server services. The database upgrade scripts are run from the “server” 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
server	python	The center of the Peek Platform, ideal for central logic.
storage	python	This refers to support for persisting and retrieving database data.
client	python	The client service handles requests from 'desktop' and 'mobile'.
agent	python	The agent is a satellite service, integrating with external systems.
worker	python	The worker service provides parallel processing for computational intensive tasks
admin	typescript	A web based admin interface for the peek platform
mobile	typescript	The user interface for mobile devices.
desktop	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 Server Service

The Peek Server Service is the central / main / core server 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 Server services.

2.2.3 Client Service

The Client service was introduced to handle all requests from desktop, mobile and web apps. Reducing the load on the Peek Server.

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

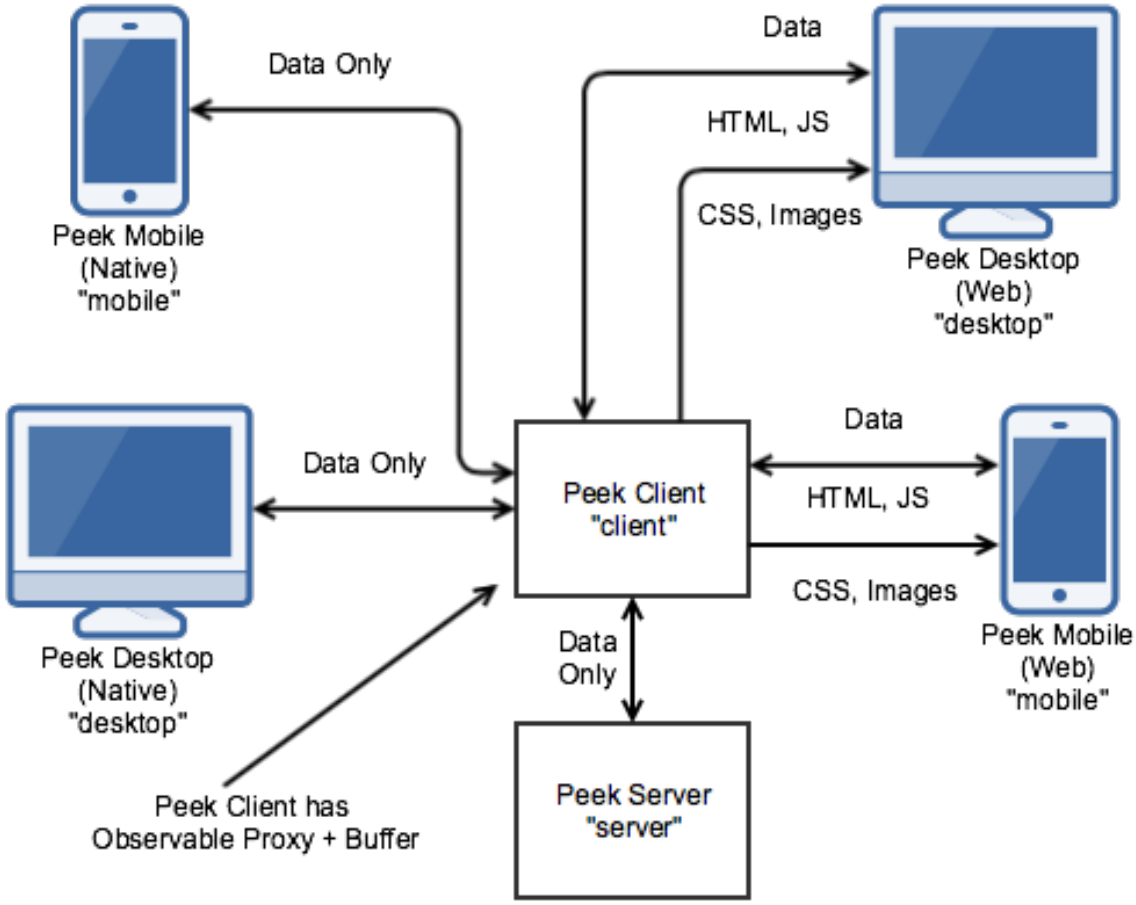
The Peek Client server handles all the live data, and serves all the resources to the Peek Desktop and Peek Mobile services.

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

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

The Client serves all HTTP resources to the Desktop web apps and Mobile web apps, this includes HTML, CSS, Javascript, images and other assets.

The following diagram gives an overview of the clients communications.



2.2.4 Mobile Service

The mobile service provides two user interfaces, a native mobile app backed by Telerik Nativescript + Angular, and an Angular web app.

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

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

In web developers terminology, the Mobile service is called the frontend, and the Client service is called the backend.

The Mobile service codes structure allows Angualar components to be reused to drive both nativescript and web based interfaces. For example :

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

2.2.5 Desktop Service

The Peek Desktop service is almost identical to the Mobile service, using Electron + Angular for Native desktop apps and Angular for the web app.

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

The Desktop service codes structure allows Angualar components to be reused to drive both electron and web based interfaces. For example :

- **my-component.tron.html** (View for Nativescipt 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 Mobile and Desktop services if they choose.

2.2.6 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 Server 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 achives better performance.

Multiple Peek Worker services can connect to one Peek Server service.

2.2.7 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 Server service.

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

Here are some example use cases :

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

2.2.8 Admin Service

The Peek Admin service is the Peek Administrators user interface, providing administration for plugins and the platform.

The Peek Admin service is almost identical to the Desktop service, 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 it's self. 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
server	YES
storage	no
client	YES
agent	YES
worker	no
admin	YES
mobile	YES
desktop	YES

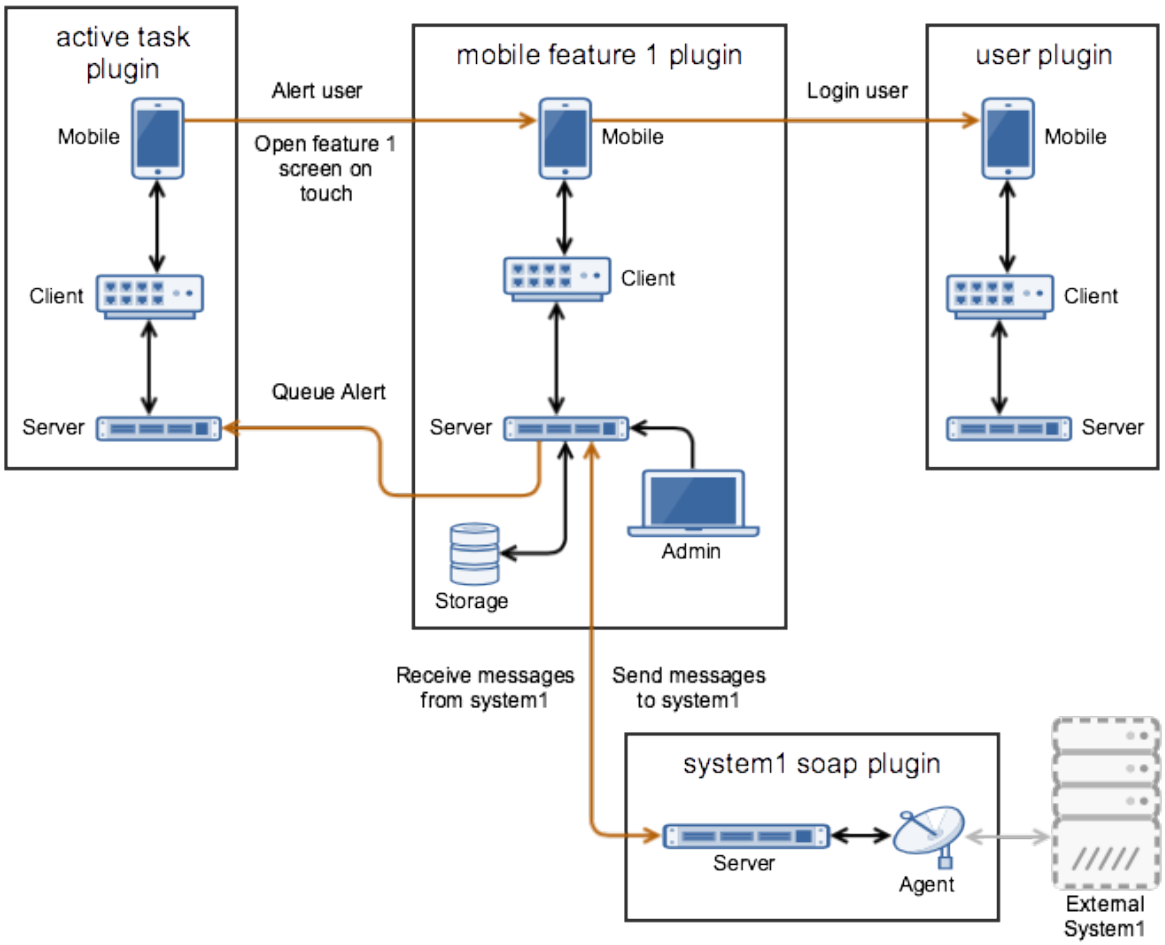
You could create other "User Plugins" with the same exposed plugin API for different backends, and the "Active Task" plugin wouldn't know the difference.

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 system1. It handles the burdon of implementing the system 1 SOAP interface.
- The SOAP, User and Active Task plugins provide APIs on the server 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

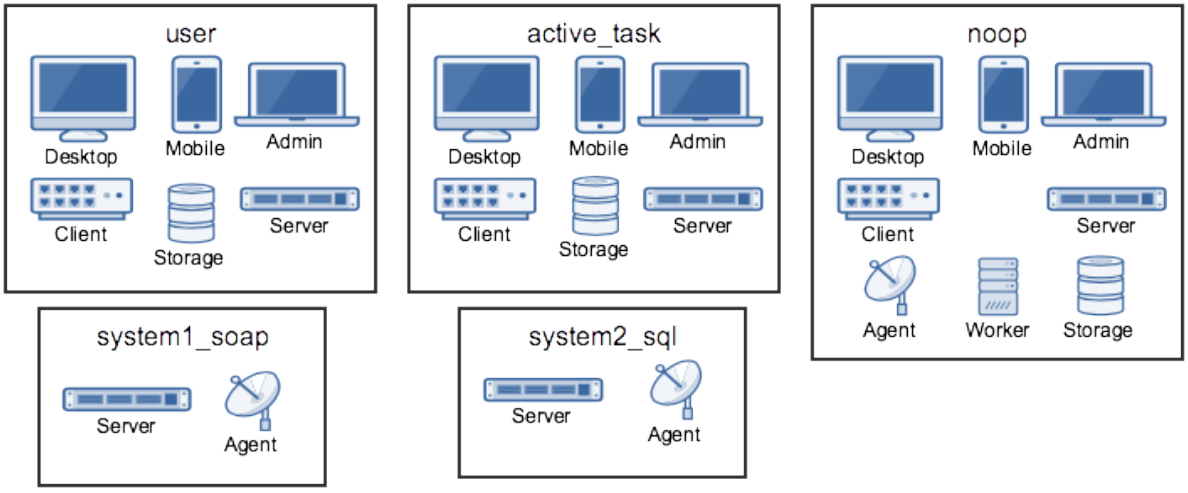
All of the code for one plugin exists within a single python package. This one package is installed on all of the services, even though only part of the plugin will run on each service.

There are multiple entry hooks with in 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 server service, and part of the plugin may run on the agent service.

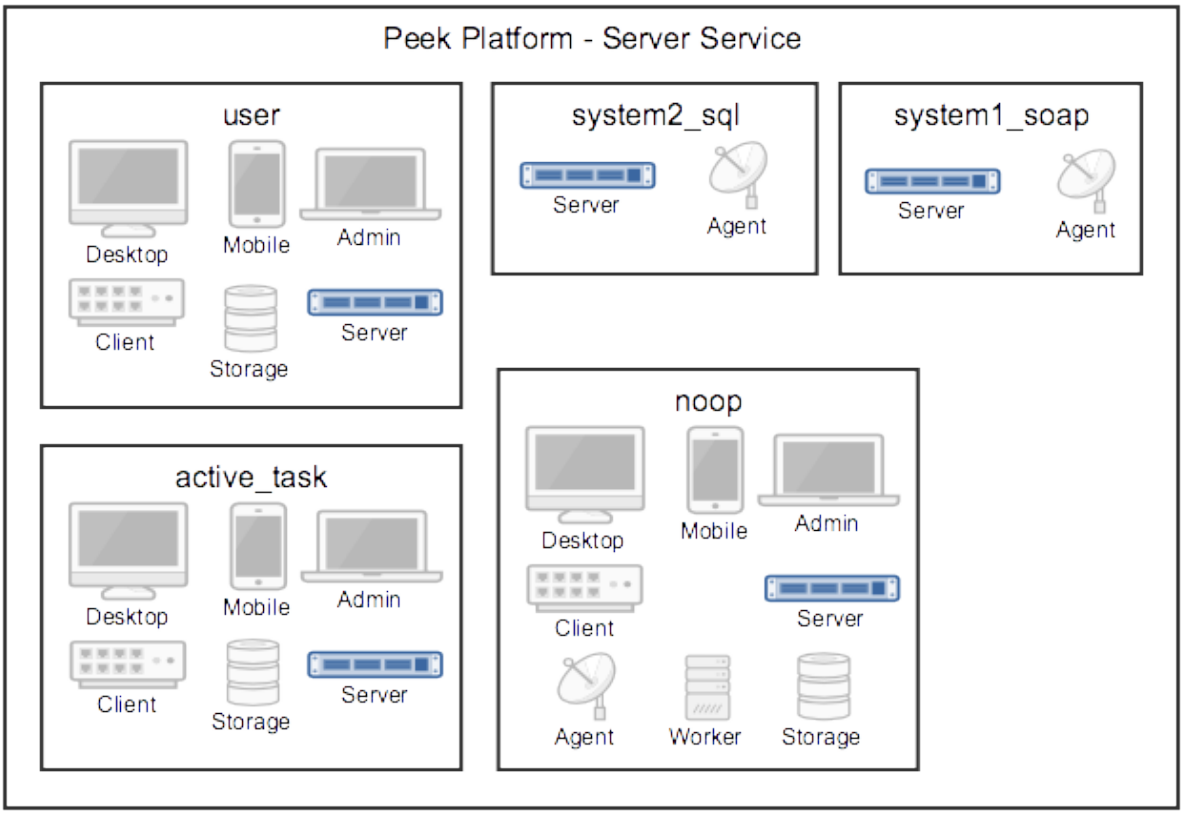
Here are some plugin examples, indicating the services each platform has been designed to run on. Here are some things of interest :

- 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 server service, this is an ideal place for plugins to integrate with each other.

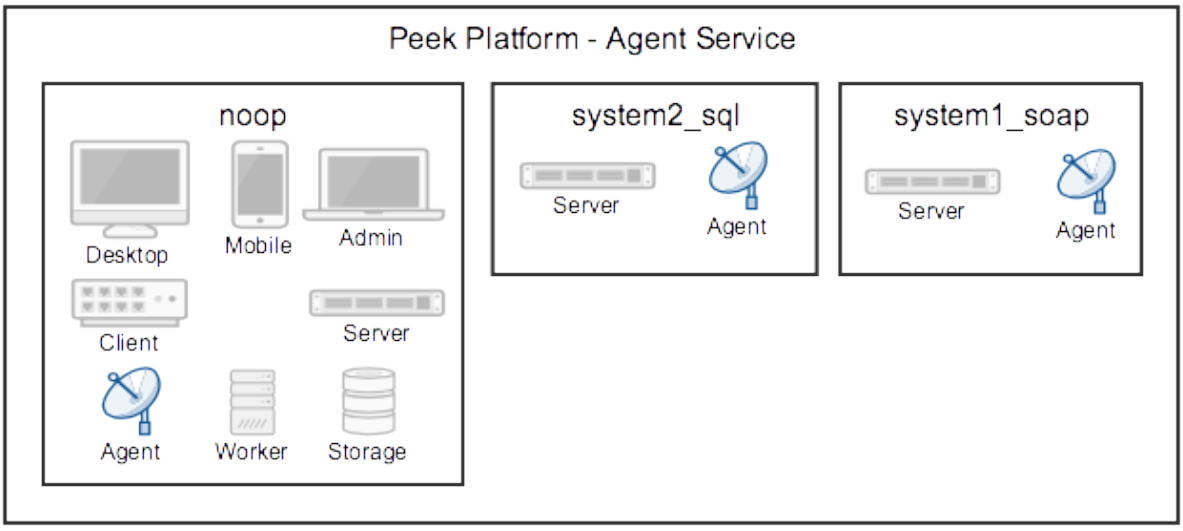


This diagram illustrates how the plugins will run on the server service.

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



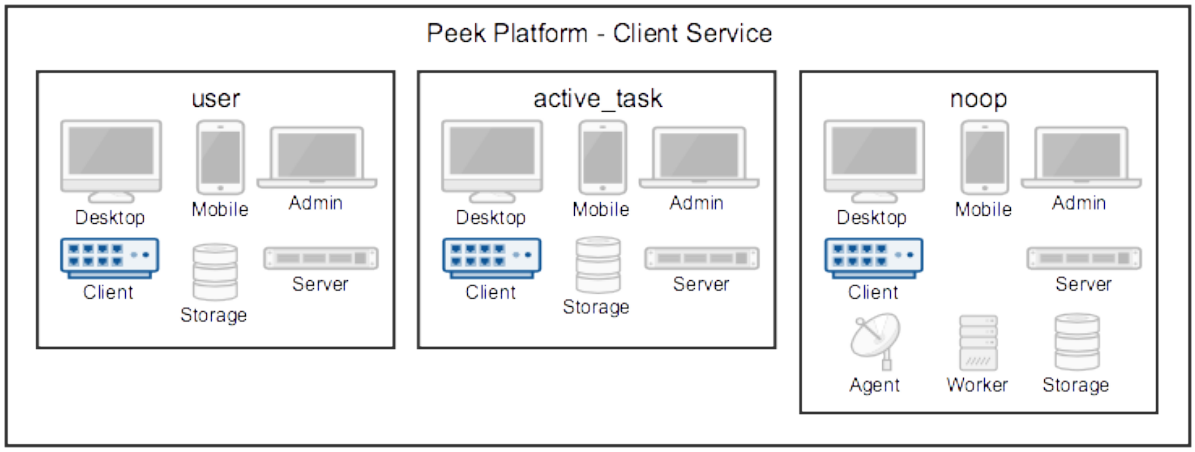
There are only two plugins that require the agent service, so the agent will only load these two. Again, the whole plugin is installed in the agents python environment.



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

The client, agent, worker and server services can and run from the one python environment. This is the standard setup

for single server environments.

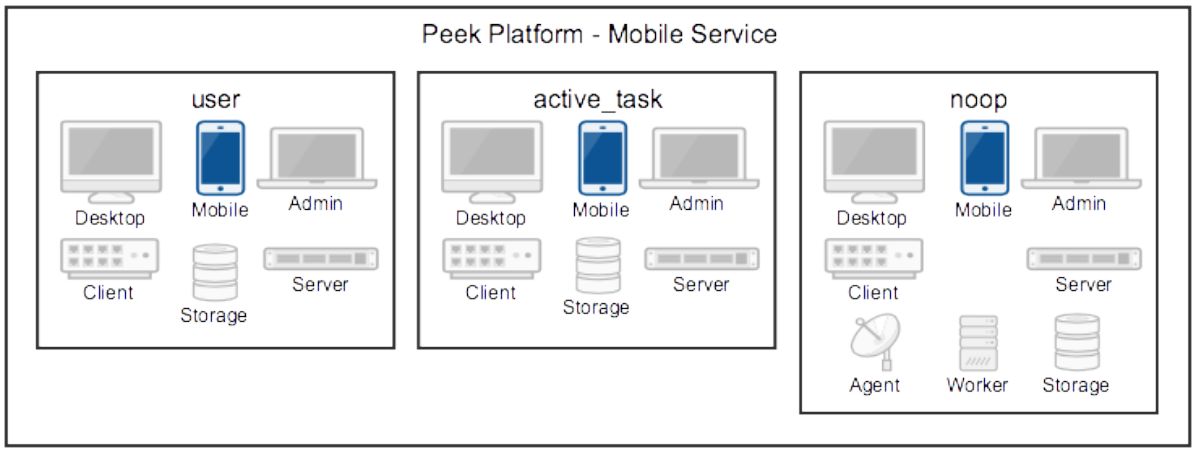


There are three plugins that require the mobile service. The mobile service is a python package that contains the build skeletins for the nativescript and web apps.

The client service combines (copies) the files required from each of the plugins into the build environments, and then compiles the web app. (The Nativescript app is compiled manually by developers)

The client and server services prepare and compile the desktop, mobile and admin services, as these are all HTML, Typescript and Nativescript.

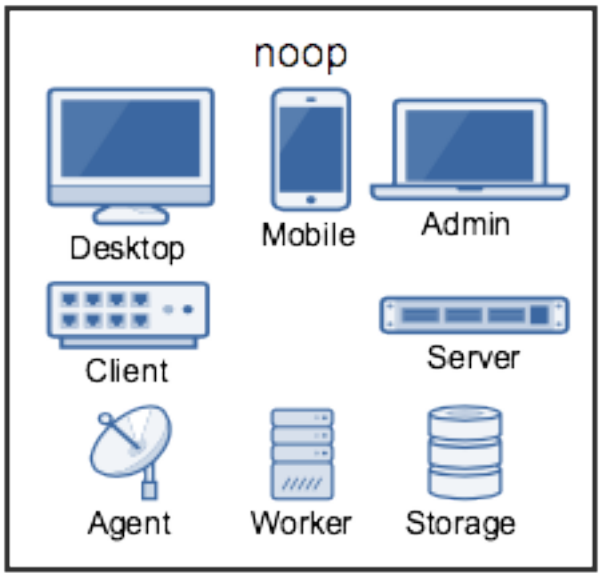
The desktop, mobile and admin interfaces need the client and server python services to run, so this compile arrangement makes sense.



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)
      - `admin-app` (The admin web based user interface)
      - `admin-assets` (Static assets for the admin web UI)
      - `agent` (The code that runs on the agent service)
      - `alembic` (Database schema versioning scripts)
      - `client` (The code that runs on the client service)
      - `desktop-app` (The user interface that runs on the desktop/web)
      - `desktop-assets` (Images for the desktop/web)
      - `mobile-app` (The user interface that runs on the mobile/web devices)
      - `mobile-assets` (Images for the mobile/web UI)
      - `server` (The code that runs on the server service)
      - `storage` (SQLAlchemy ORM classes for db access, used by server,worker)
      - `tuples` (Private data structures)
      - `worker` (The parallel processing Celery tasks that are run on the worker)
    - `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)
      - `desktop` (Exposed API, index.ts exposes desktop only declarations)
      - `mobile` (Exposed API, index.ts exposes mobile only declarations)
      - `admin` (Exposed API, index.ts exposes admin only declarations)

        - `_private` (Code only used by this plugin)
        - `desktop` (Private desktop declarations)
        - `mobile` (Private mobile declarations)
        - `admin` (Private admin declarations)
    * `agent` (Exposed API, plugins on the agent service use this)
    * `client` (Exposed API, plugins on the client service use this)
    * `server` (Exposed API, plugins on the server 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 hypons in them?

CHAPTER 3

Setup OS Requirements

3.1 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.

3.1.1 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
- PostgresSQL 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*.

3.1.2 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"
```

3.1.3 Installation Guide

The following sections begin the installation procedure.

3.1.4 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$$W0RD

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.

3.1.5 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.

3.1.6 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

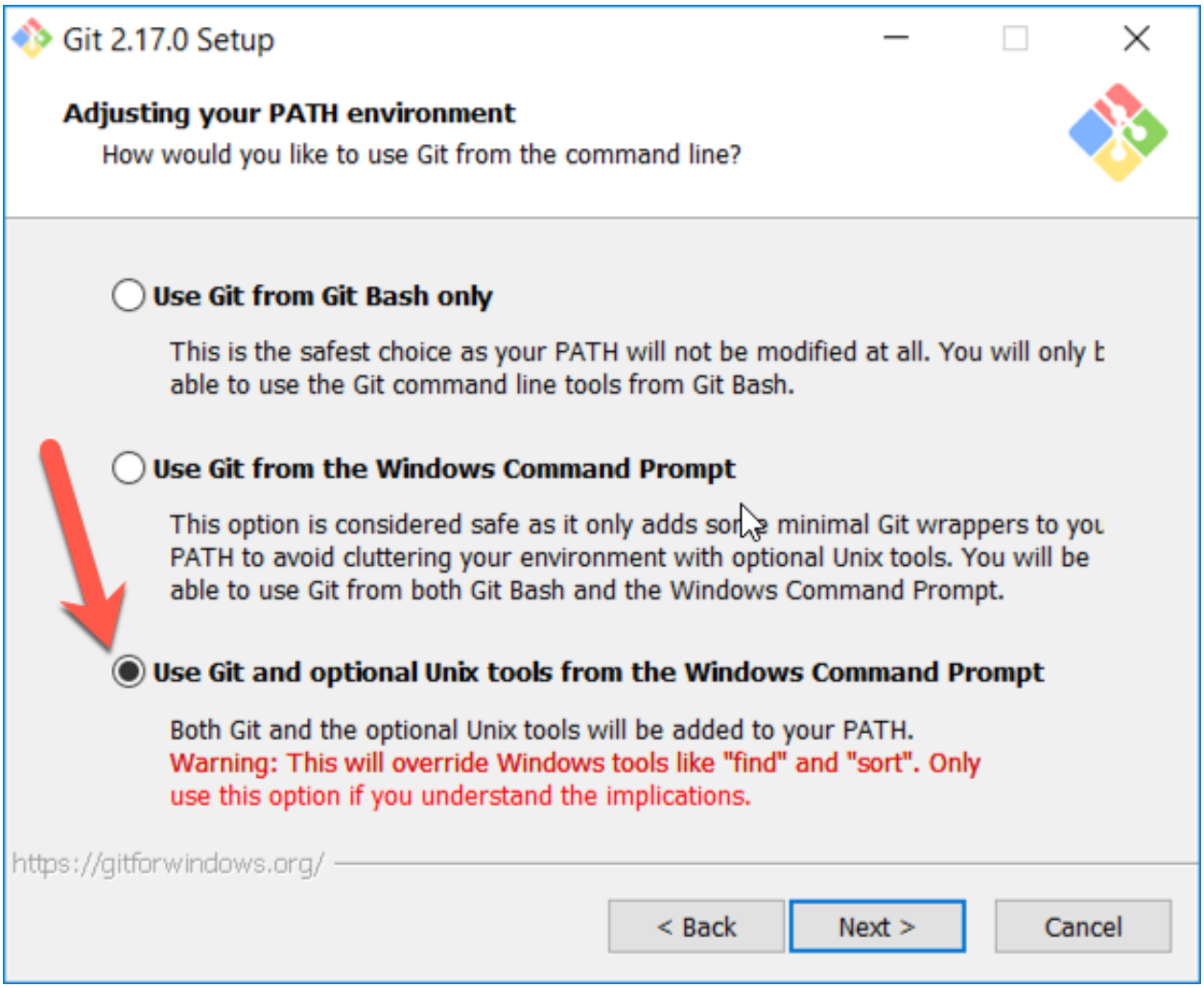
3.1.7 Setup Msys Git

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.

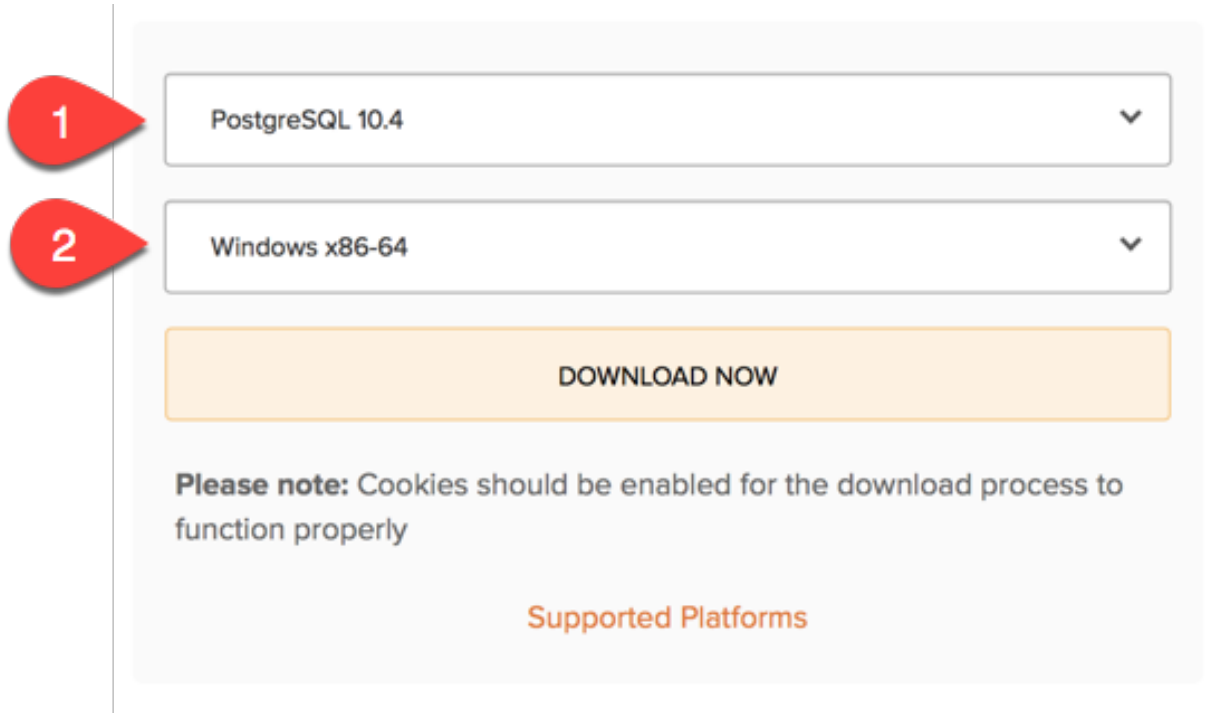
3.1.8 Install PostgresSQL

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")



Install PostgresSQL with default settings.

Make a note of the postgres user password that you supply, you'll need this.

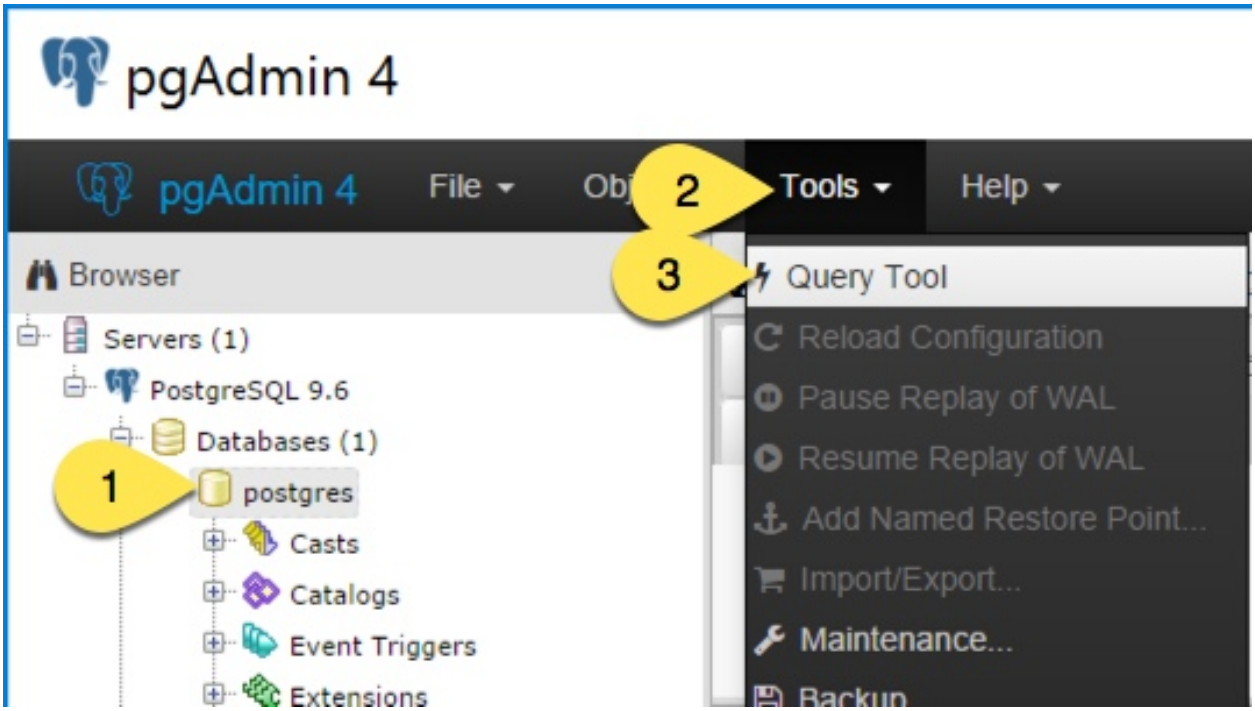
Warning: Generate a strong password for both peek and postgres users for production use.

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

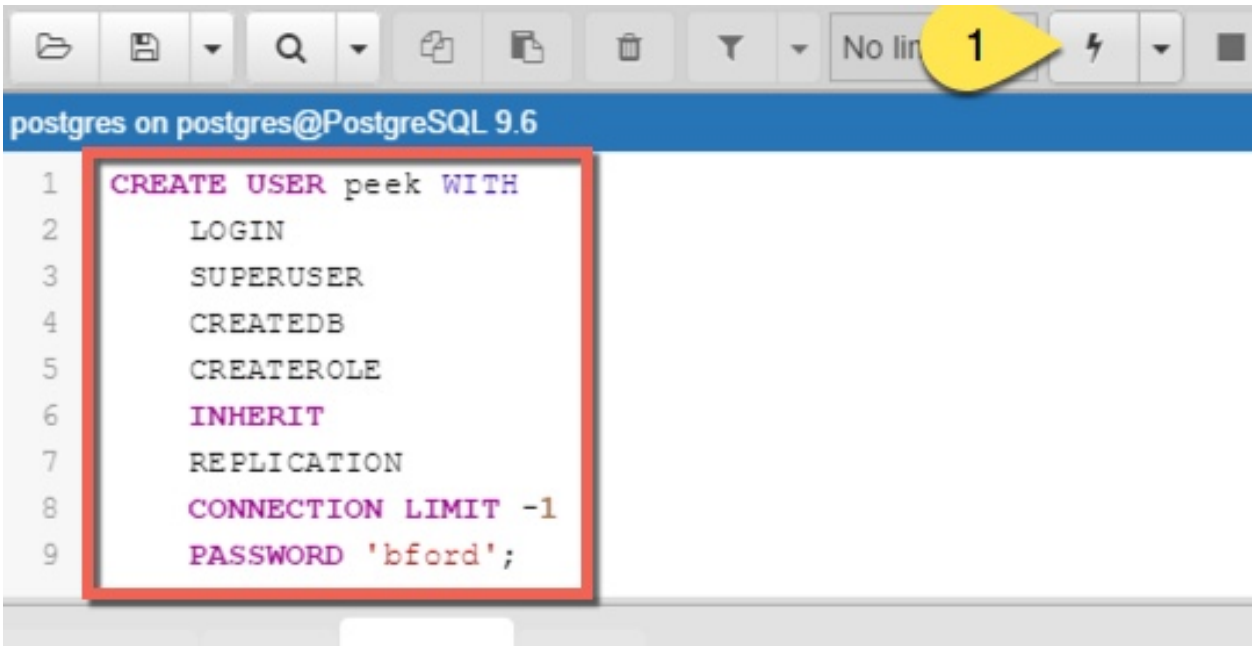


Create the peek user, run the following script:

```
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.

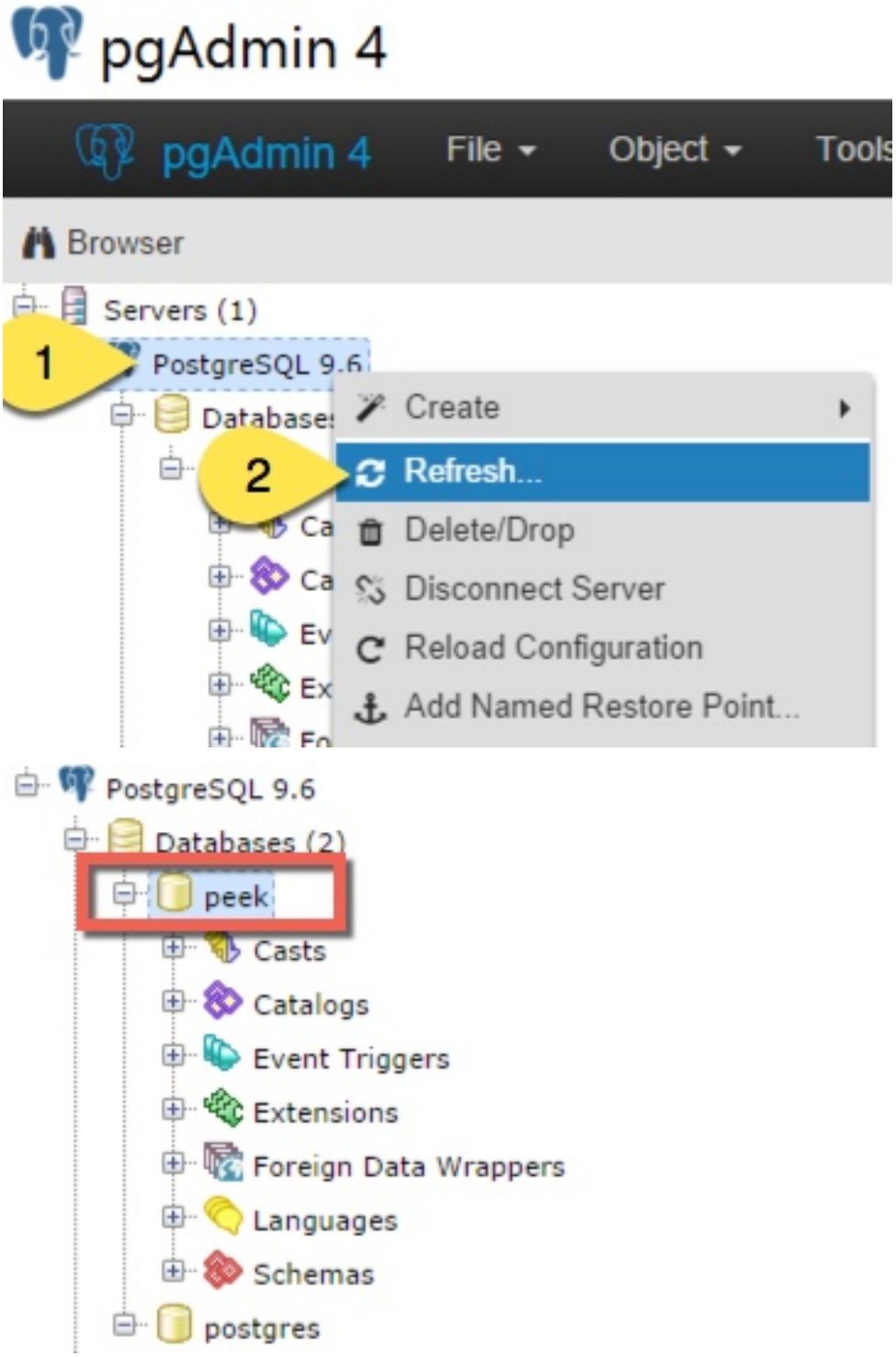
Example:



Create the peek database, run the following script:

```
CREATE DATABASE peek WITH
    OWNER = peek
    ENCODING = 'UTF8'
    CONNECTION LIMIT = -1;
```

Confirm database was created

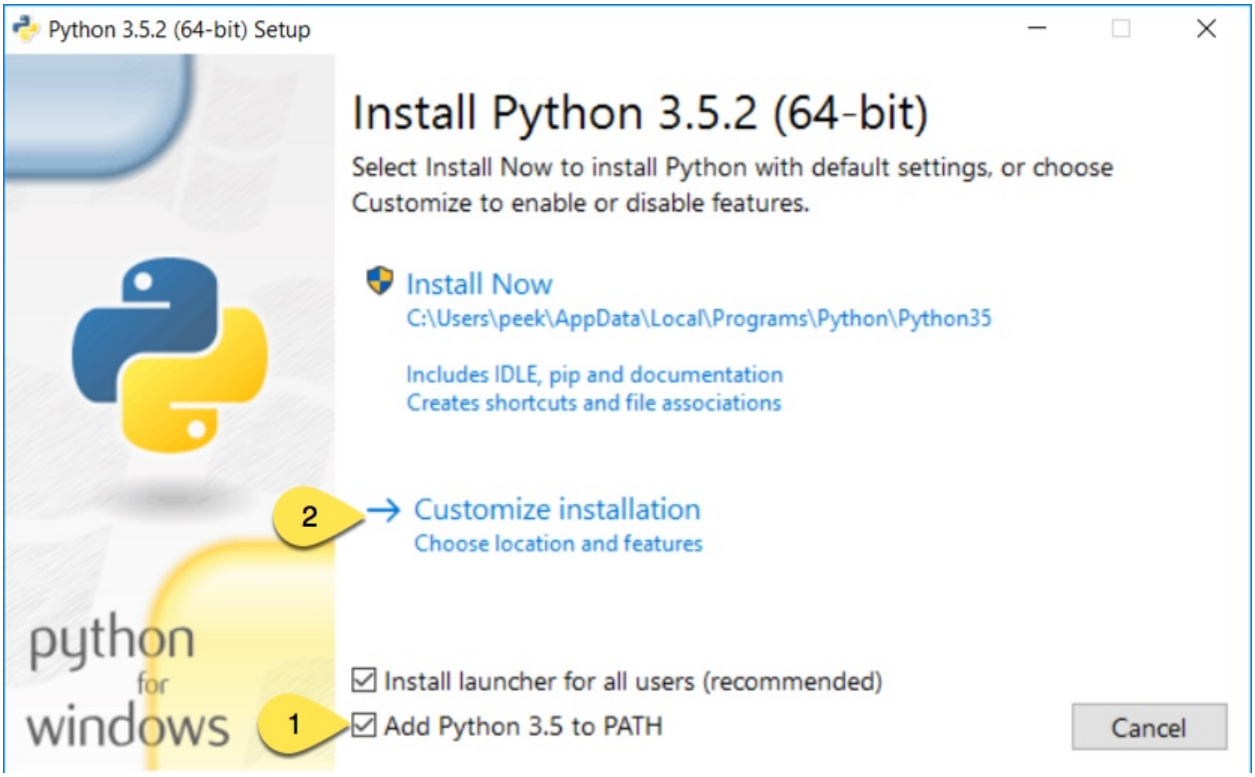


3.1.9 Install Python 3.6

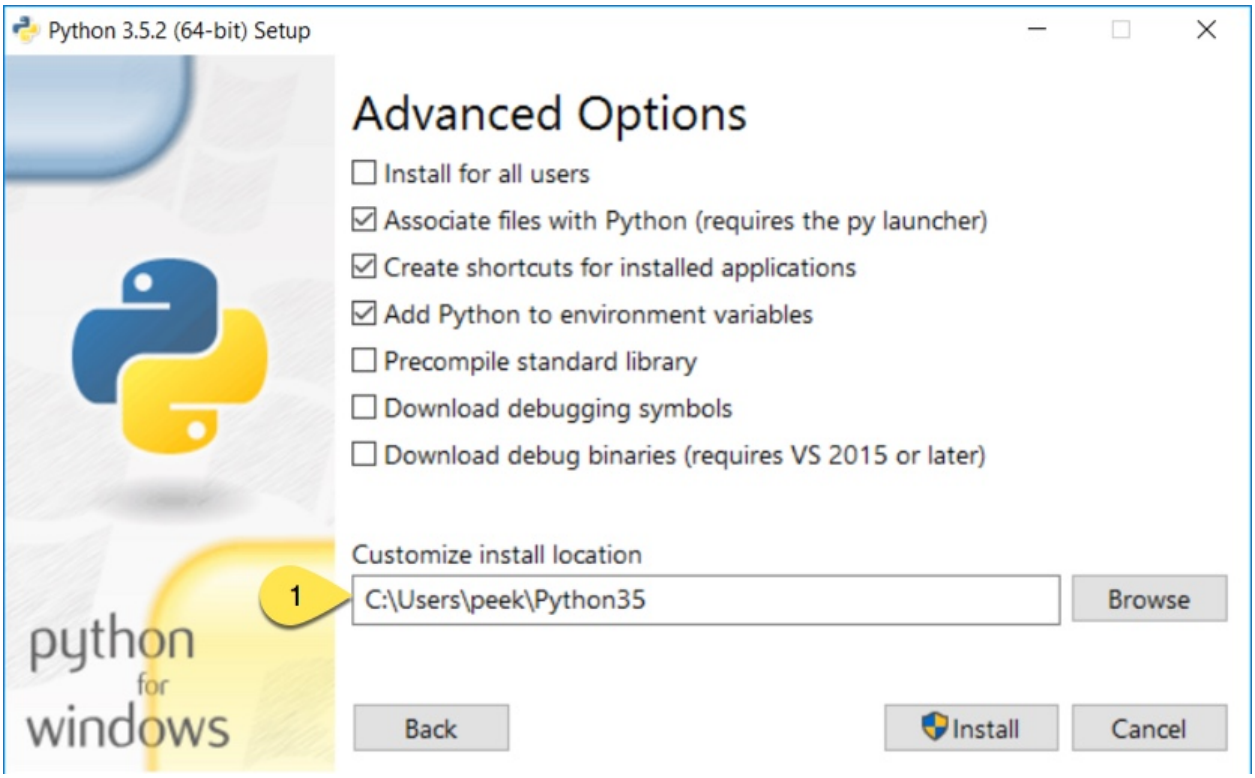
Download https://www.python.org/ftp/python/3.6.8/python-3.6.8-amd64.exe

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
```

3.1.10 Install Worker Dependencies

Install the parallel processing queue we use for the peek-worker 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
```

3.1.11 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.

3.1.12 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

3.1.13 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

3.2 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.

3.2.1 Installation Objective

This Installation Guide contains specific Mac 10.12 Sierra 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)
- Oracle JDK
- Homebrew
- Python 3.6.x
- Postgres 10.3

Optional Software

- Oracle 12c 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.

3.2.2 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.

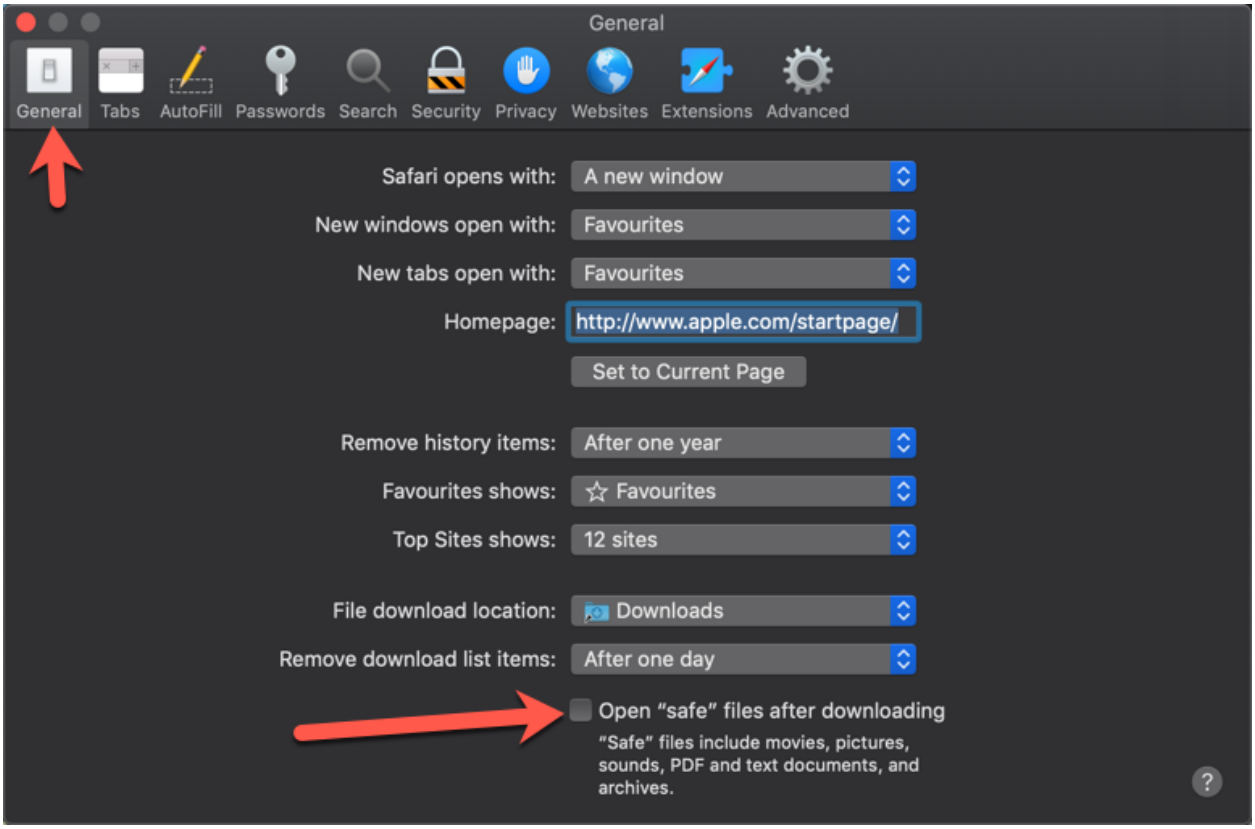
3.2.3 Safari Open Safe Files

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.

3.2.4 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$$W0RD

Sign in to the peek account.

Important: All steps after this point assume you're logged in as the peek user.

3.2.5 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:

```
xcode-select --install
```

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'

3.2.6 Install an Oracle JDK

Download the macOS disk image:

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

3.2.7 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
```

3.2.8 Install PostGreSQL

Install the relational database we use on macOS.

In terminal run:

```
brew install postgresql
```

Start postgresql and create start at login launchd service:

```
brew services start 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
# 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 Postgres user

```
createuser -d -r -s peek
```

Create the database

```
createdb -O peek peek
```

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 databast administration tool.

Download the latest version of pgAdmin4 for macOS from the following link

https://www.pgadmin.org/download/pgadmin-4-macos/

3.2.9 Install Python 3.6

Edit `~/.bash_profile` and insert the following:

```
##### SET THE PEEK ENVIRONMENT #####
# Setup the variables for PYTHON
export PEEK_PY_VER="3.6.8"
export PATH="/Users/peek/cpython-${PEEK_PY_VER}/bin:$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.

Download and unarchive the supported version of Python

```
cd ~
source .bashrc
wget "https://www.python.org/ftp/python/${PEEK_PY_VER}/Python-${PEEK_PY_VER}.tgz"
tar xzf Python-${PEEK_PY_VER}.tgz
```

Configure the build

```
cd Python-${PEEK_PY_VER}

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=/Users/peek/cpython-${PEEK_PY_VER}/ --enable-optimizations
```

Make and Make install the software

```
make install
```

Cleanup the download and build dir

```
cd
rm -rf Python-${PEEK_PY_VER}*
```

Symlink the python3 commands so they are the only ones picked up by path.

```
cd /Users/peek/cpython-${PEEK_PY_VER}/bin
ln -s pip3 pip
ln -s python3 python
cd
```

Open a new terminal and test that the setup is working

```
pass="/Users/peek/cpython-3.6.8/bin/python"
[ "`which python`" == "$pass" ] && echo "Success" || echo "FAILED"

pass="Python 3.6.8"
[ "`python --version`" == "$pass" ] && echo "Success" || echo "FAILED"

pass="/Users/peek/cpython-3.6.8/bin/pip"
[ "`which pip`" == "$pass" ] && echo "Success" || echo "FAILED"


pass="pip 18.1 from /Users/peek/cpython-3.6.8/lib/python3.6/site-packages/pip (python␣
↪3.6)"
[ "`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
```

3.2.10 Install Worker Dependencies

Install the parallel processing queue we use for the peek-worker 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"
```

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
```

3.2.11 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.

Make the directory where the oracle client will live

```
mkdir ~/oracle
```

Download the following from oracle.

The version used in these instructions is `18.1.0.0.0`.

Note: Oracle version 18.1 is not available for macOS.

---

1. Download the "Basic Package" from http://www.oracle.com/technetwork/topics/intel-macsoft-096467.html
2. Download the "SDK Package" from http://www.oracle.com/technetwork/topics/intel-macsoft-096467.html

Copy these files to `~/oracle` on the peek server.

Extract the files.

```
cd ~/oracle
unzip instantclient-basic-macos.x64-18.1.0.0.0.zip
unzip instantclient-sdk-macos.x64-18.1.0.0.0.zip
```

Add links to $HOME/lib to enable applications to find the libraries:

```
mkdir ~/lib
ln -s ~/oracle/instantclient_18_1/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_18_1`"
export PATH="$ORACLE_HOME:$PATH"

##### SET THE DYLD_LIBRARY_PATH #####
export DYLD_LIBRARY_PATH="$DYLD_LIBRARY_PATH:$ORACLE_HOME"
```

3.2.12 Install FreeTDS (Optional)

FreeTDS is an open source driver for the TDS protocol, this is the protocol used to talk to the MSSQL SQLServer database.

Peek needs a installed if it uses the pymssql python database driver, which depends on FreeTDS.

Note: FreeTDS 1.x doesn't work, so be sure to install @0.91

Install FreeTDS via Homebrew:

```
brew install freetds@0.91
brew link --force freetds@0.91
```

Edit ~/.bash_profile and insert the following:

```
##### SET THE HOMEBREW ENVIRONMENT #####
# Set PATH to include fink
export PATH="/usr/local/opt/freetds@0.91/bin:$PATH"
```

Confirm the installation

```
tsql -C
```

You should see something similar to:

```
Compile-time settings (established with the "configure" script)
                             Version: freetds v0.91.112
             freetds.conf directory: /usr/local/Cellar/freetds@0.91/0.91.112/etc
     MS db-lib source compatibility: no
        Sybase binary compatibility: no
                      Thread safety: yes
                      iconv library: yes
                        TDS version: 7.1
                             iODBC: no
                          unixodbc: no
              SSPI "trusted" logins: no
                          Kerberos: no
```

3.2.13 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

3.2.14 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

3.3 Setup OS Requirements RHEL

This section describes how to perform the setup for Red Hat Linux Server 7.4. The Peek platform is designed to run on Linux.

Please read through all of the documentation before commencing the installation procedure.

3.3.1 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 3.6.x
- Postgres 10.4.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.

3.3.2 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.

3.3.3 Install Red Hat Linux Server 7.6 OS

This section installs the Red Hat Linux Server 7.6 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 7.6 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.

Staring Off

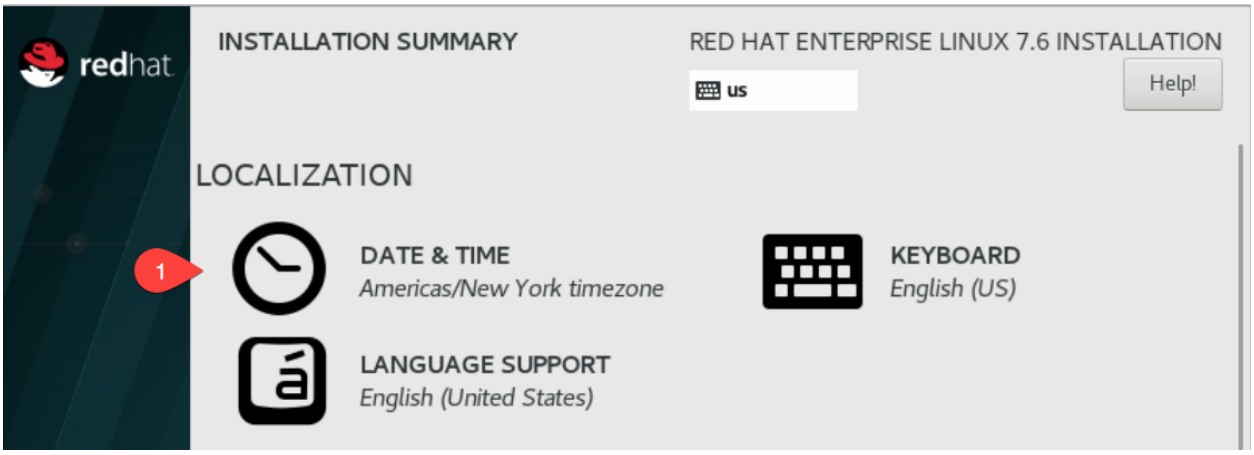
At the **Red Hat Enterprise Linux 7.6 installer boot menu** screen, select:

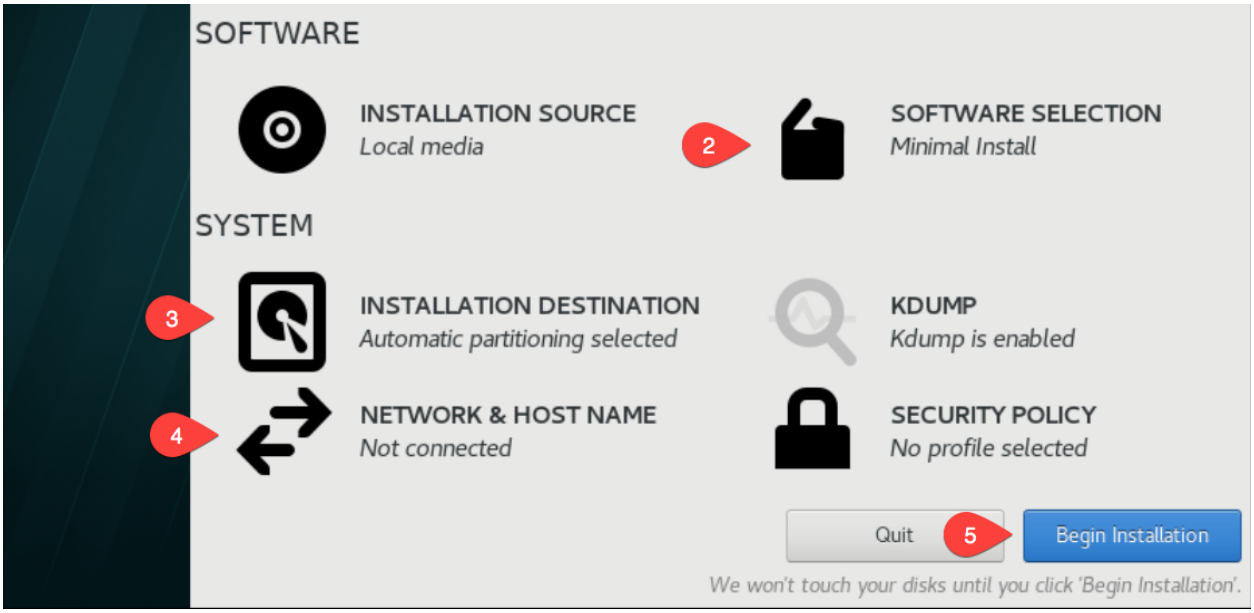
```
Install Red Hat Enterprise Linux 7.6
```

At the language selection screen, select:

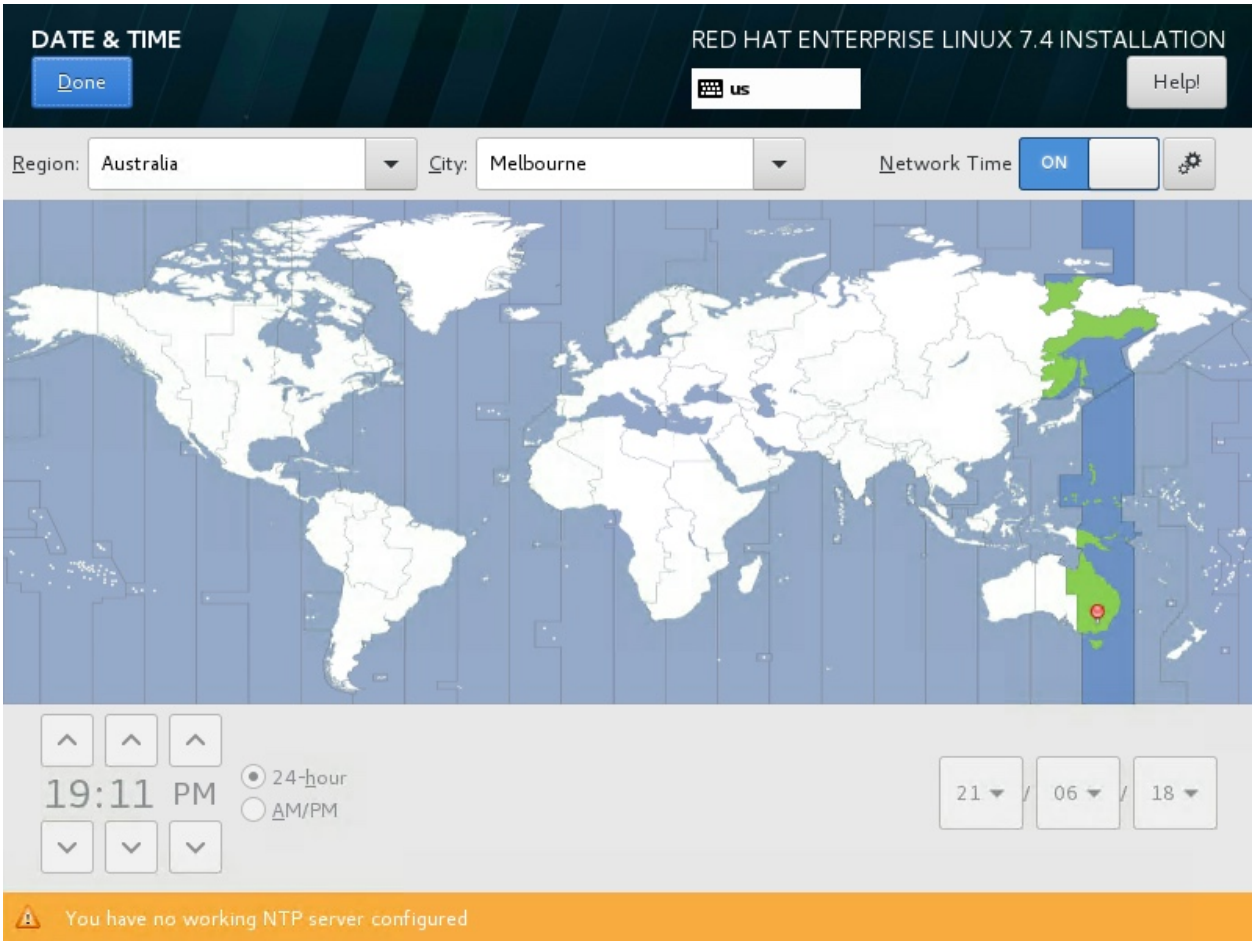
```
English
```

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.

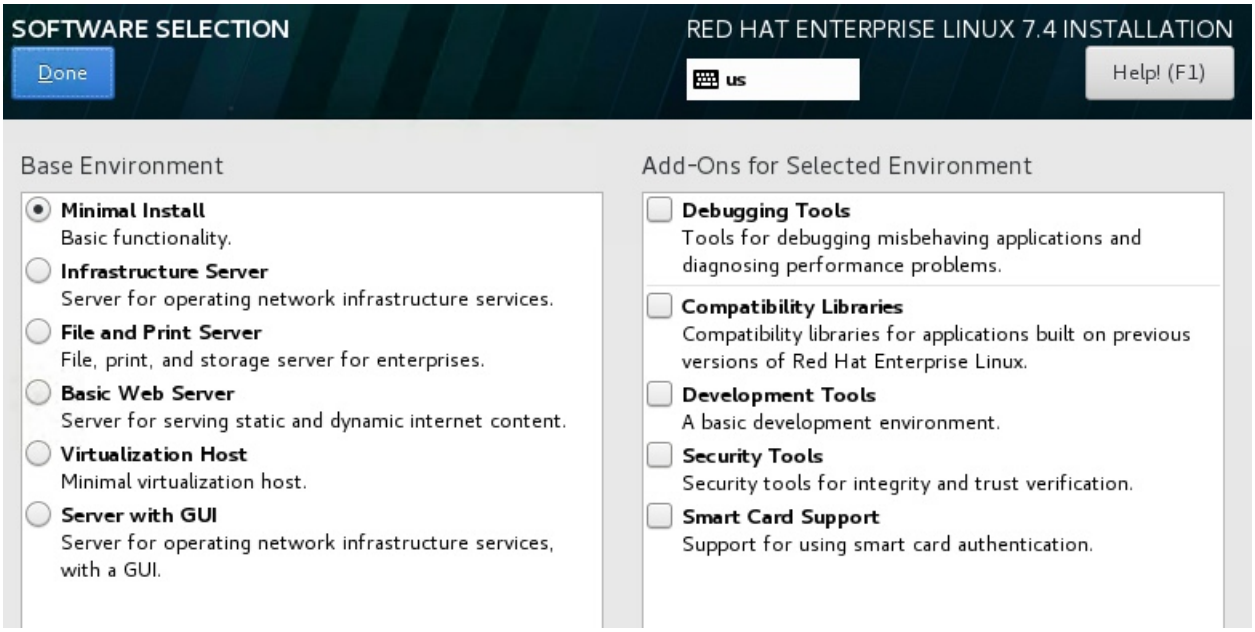




#1 Goto the **DATE & TIME** screen, select the appropriate time location.



#2 Goto the **SOFTWARE SELECTION** screen, select **Minimal Install** or **Server with GUI** if you'd like a GUI.

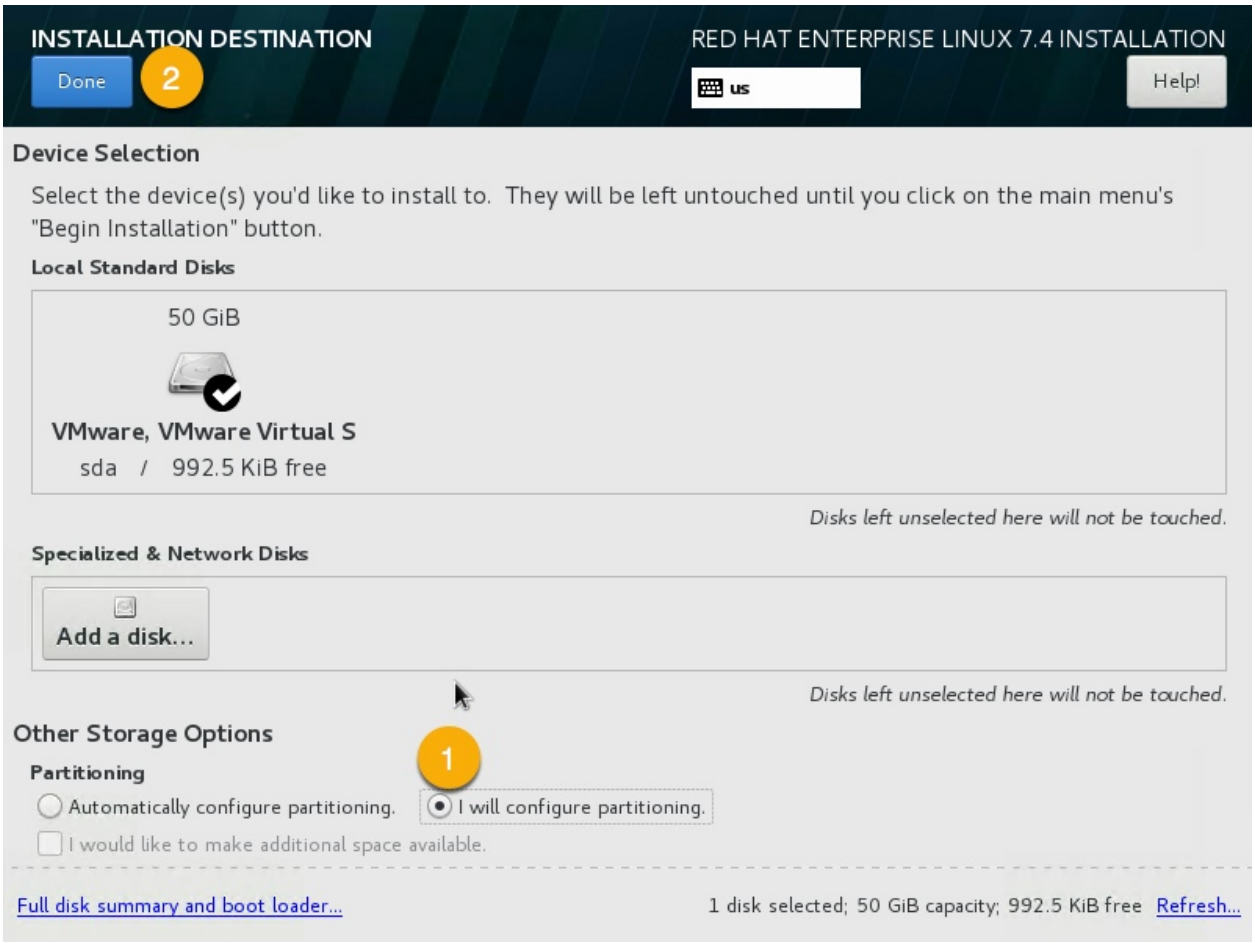


#3 Goto the **INSTALLATION DESTINATION** screen

The following partitioning is recommended for DEV peek virtual machines.

Select:

```
I will configure partitioning.
```



Select Done.

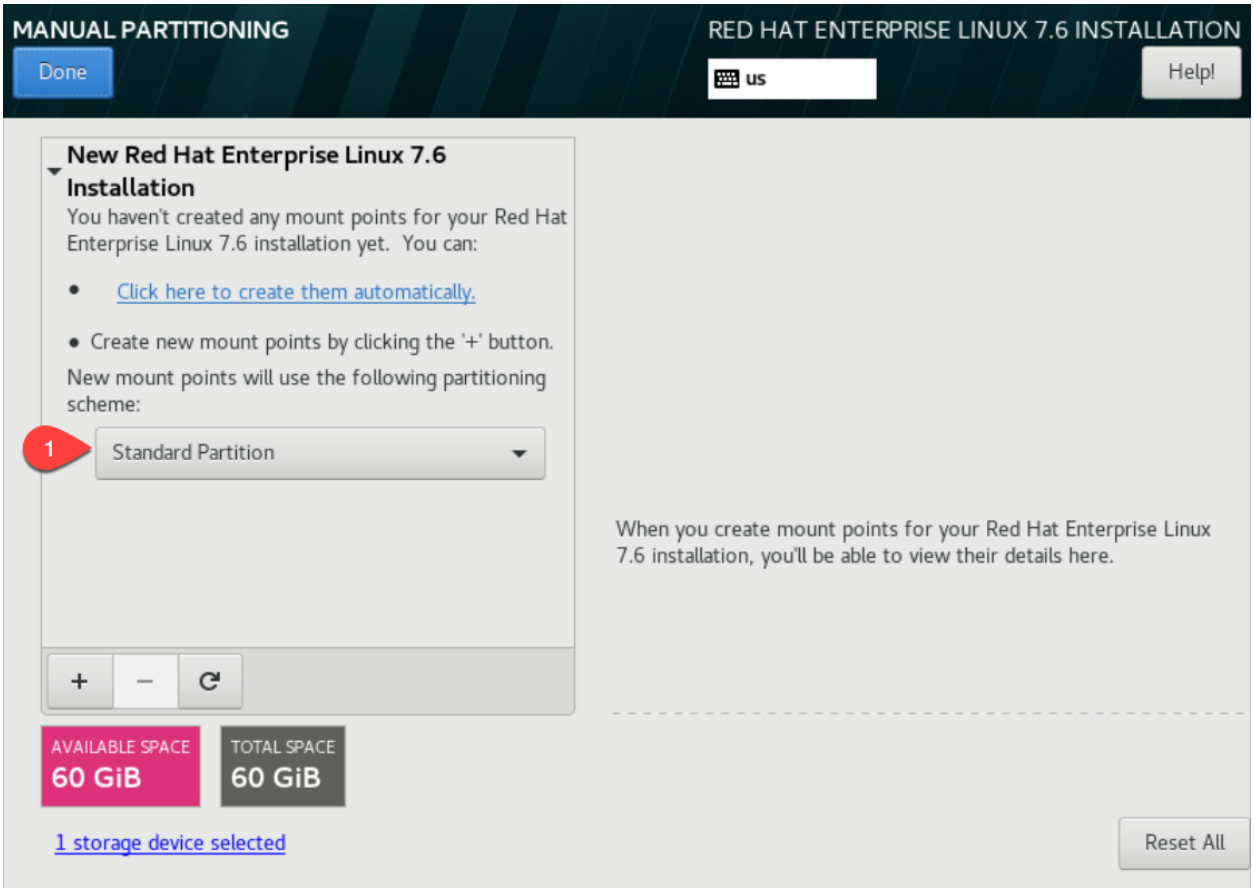
Partition Table

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.

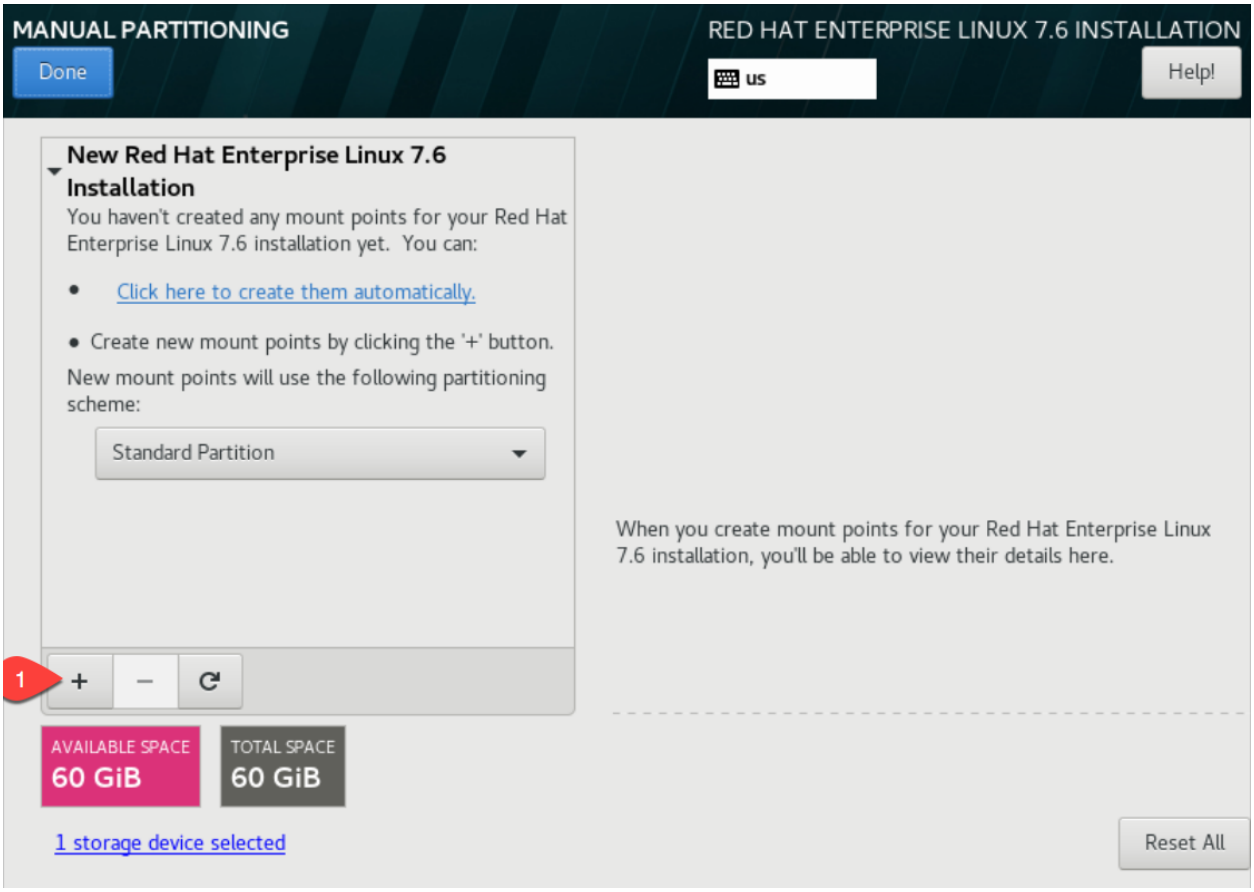
Having one file system per disk allows VM software to easily expand the disk and filesystem as required.

Select **Standard Partition**

Again, This is to allow the virtual machine software to expand the DEV server disks more easily.



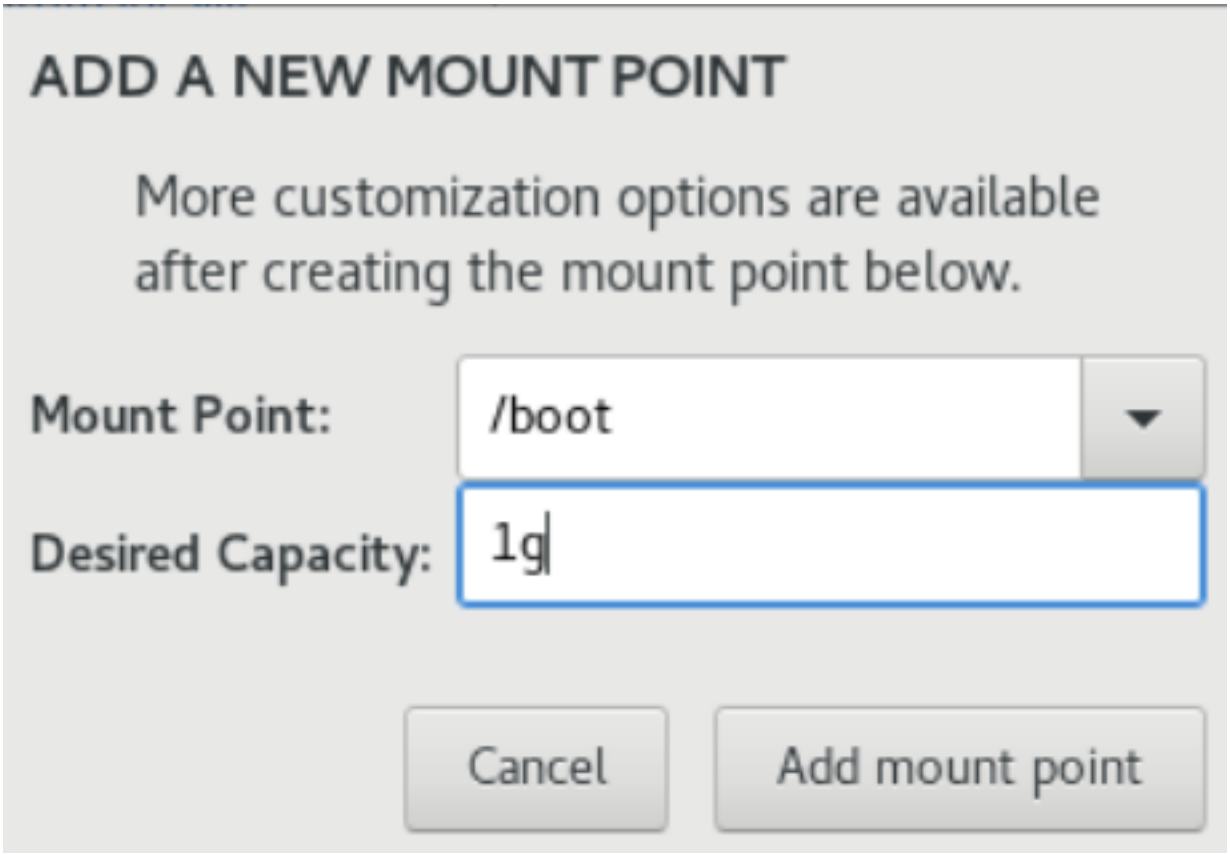
Add the partitions, for each partition, click the plus.



Set the Mount Point to **/boot**

Set the size to **1g**

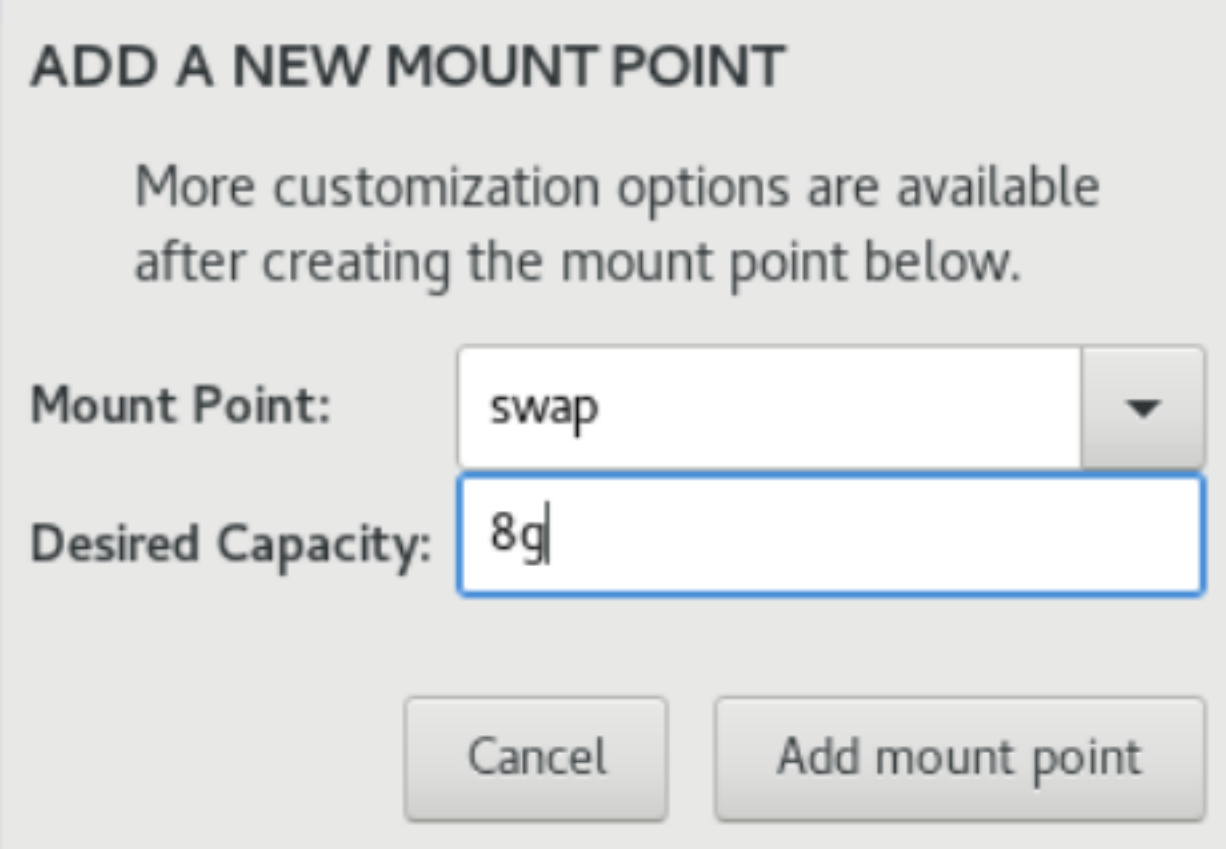
Click **Add mount point**



Set the Mount Point to **swap**

Set the size to **8g**

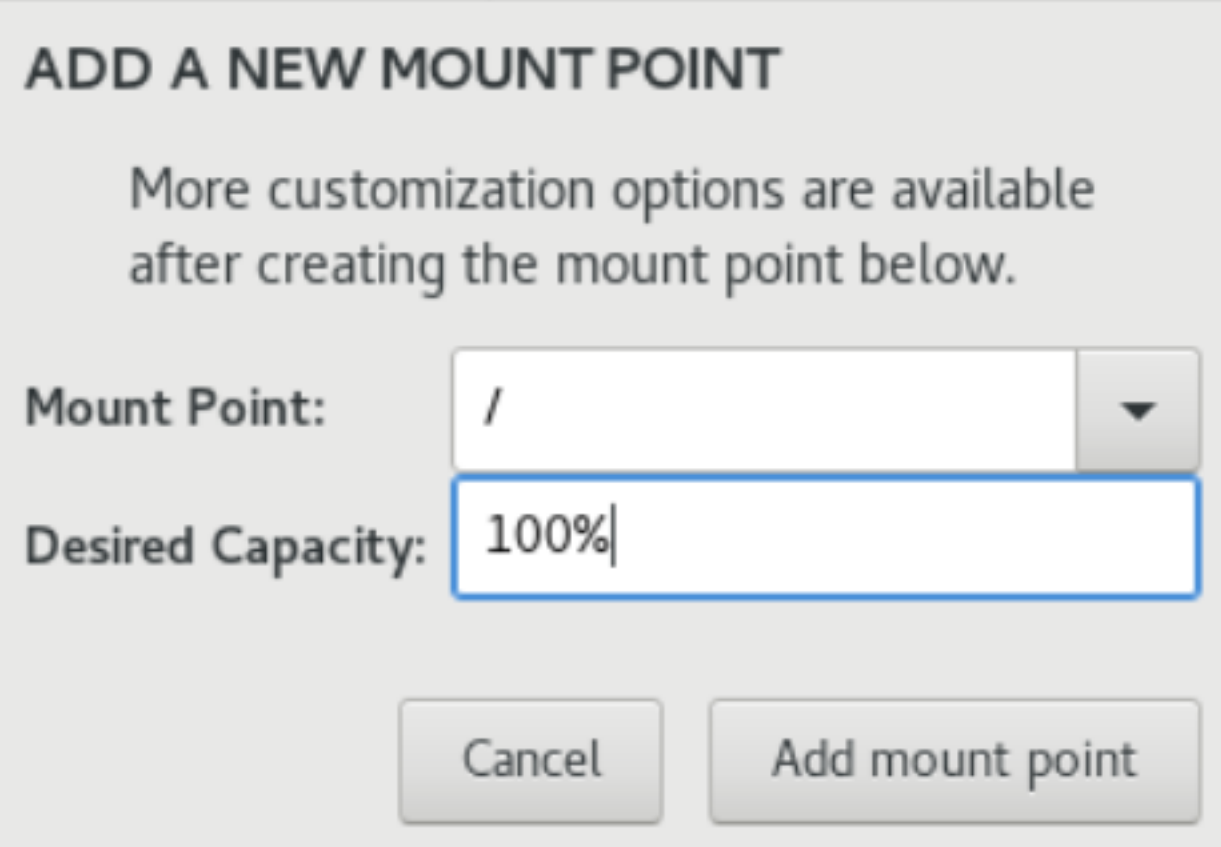
Click **Add mount point**



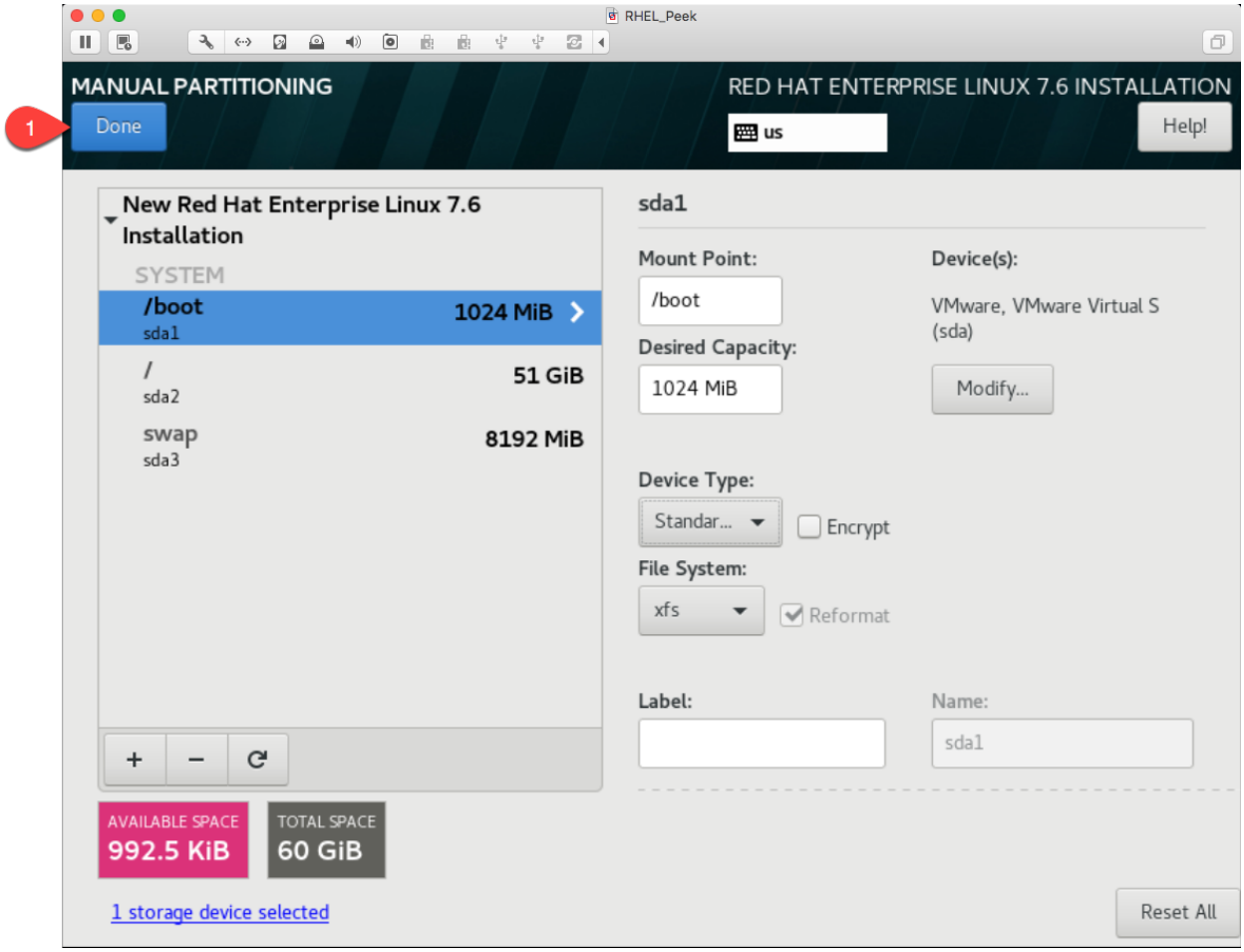
Set the Mount Point to **/**

Set the size to **100%**

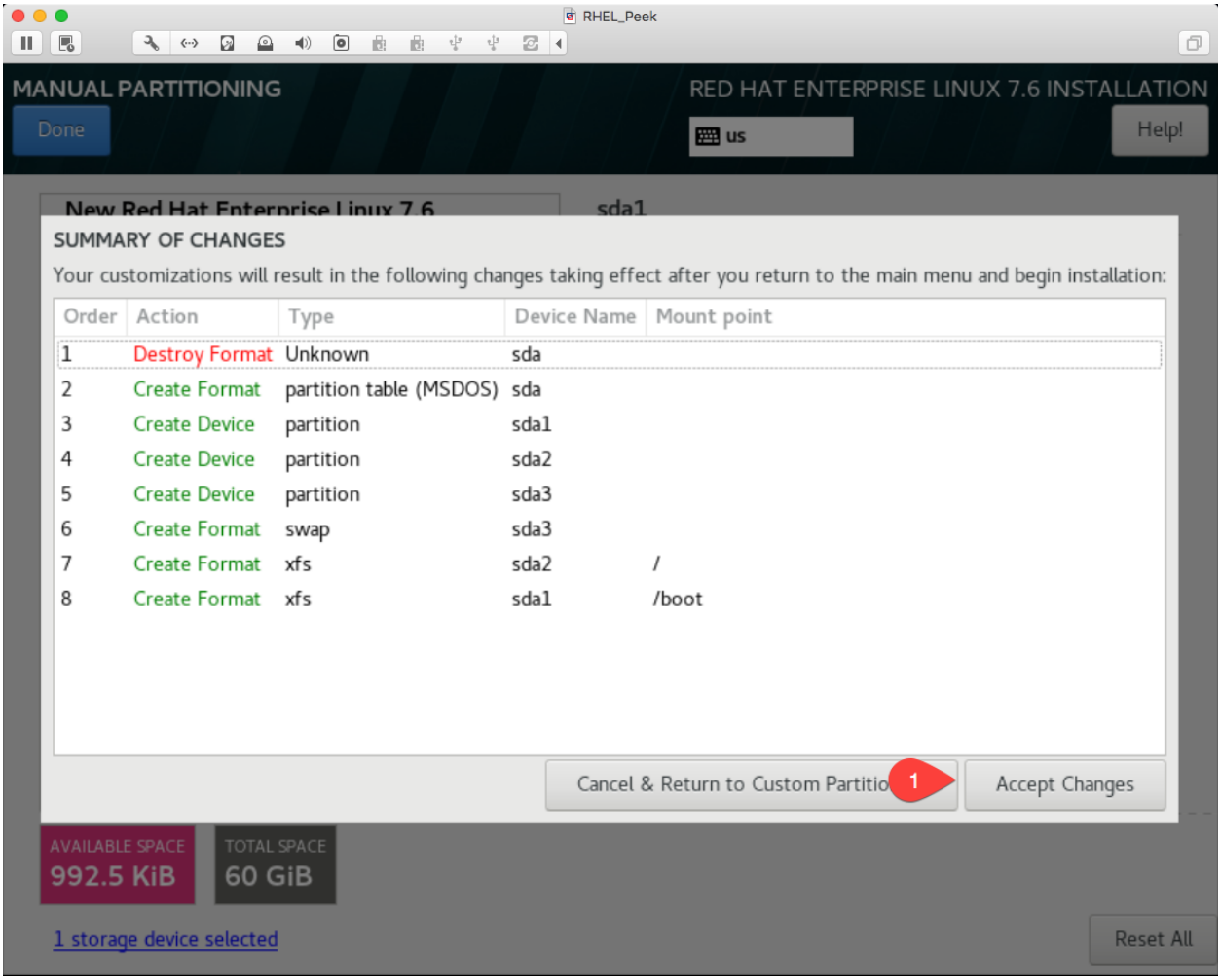
Click **Add mount point**



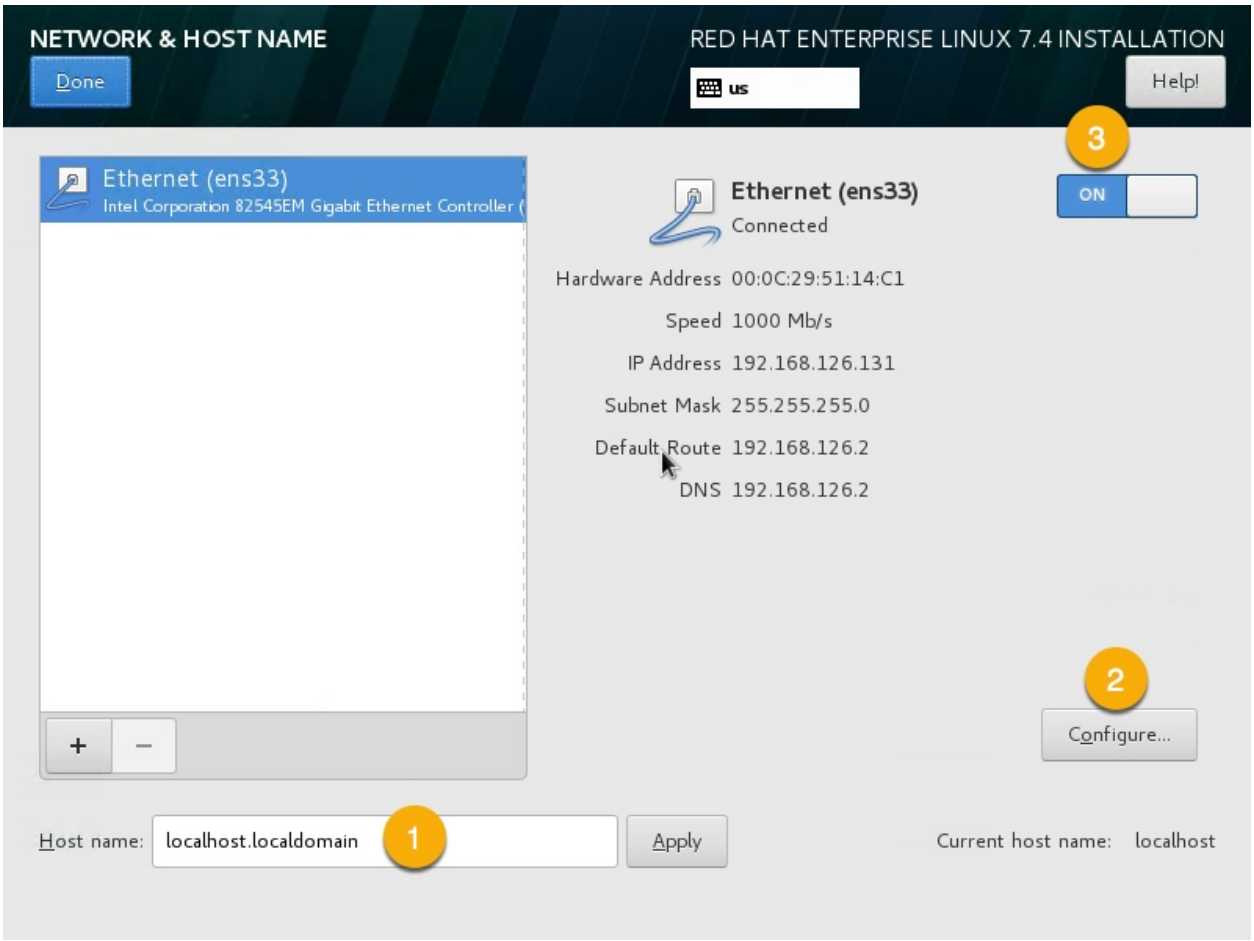
You should have a partition layout as follows, Click **Done**



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

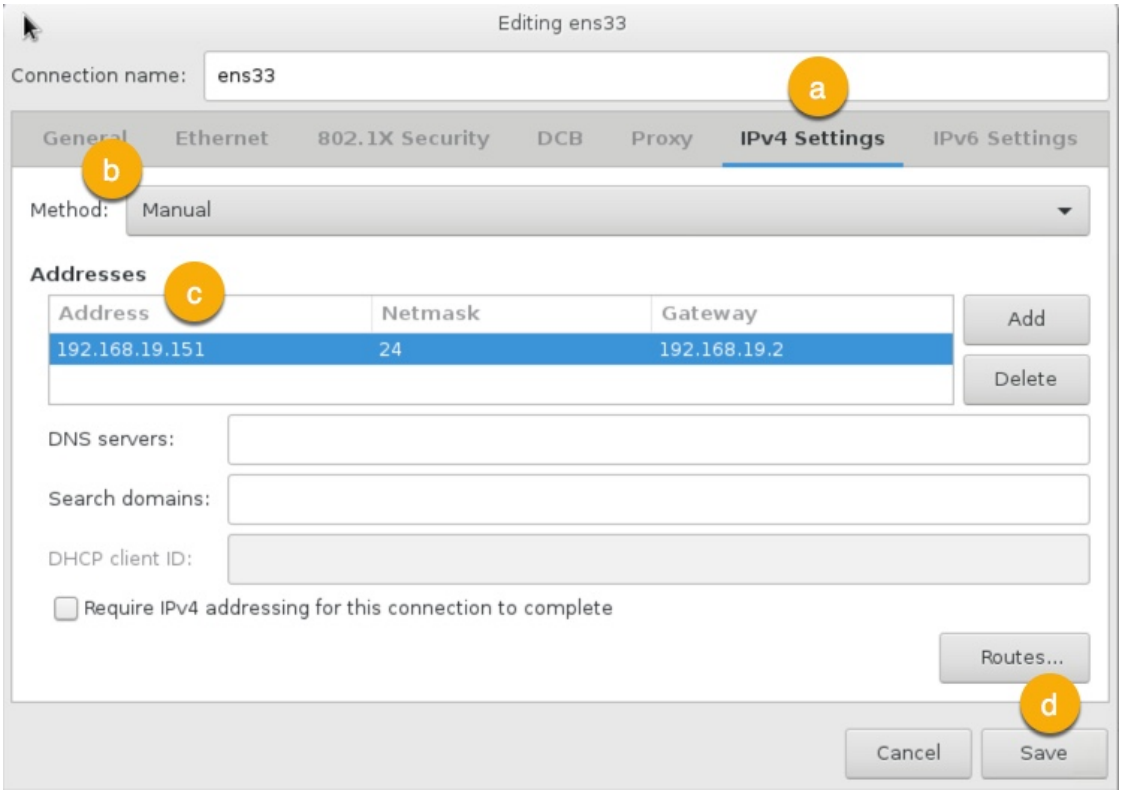
Otherwise, Configure a static IP address,

a. Goto IPv4 Settings tab,

b. Set Method to *Manual*,

c. Add static IP address,

d. Save.

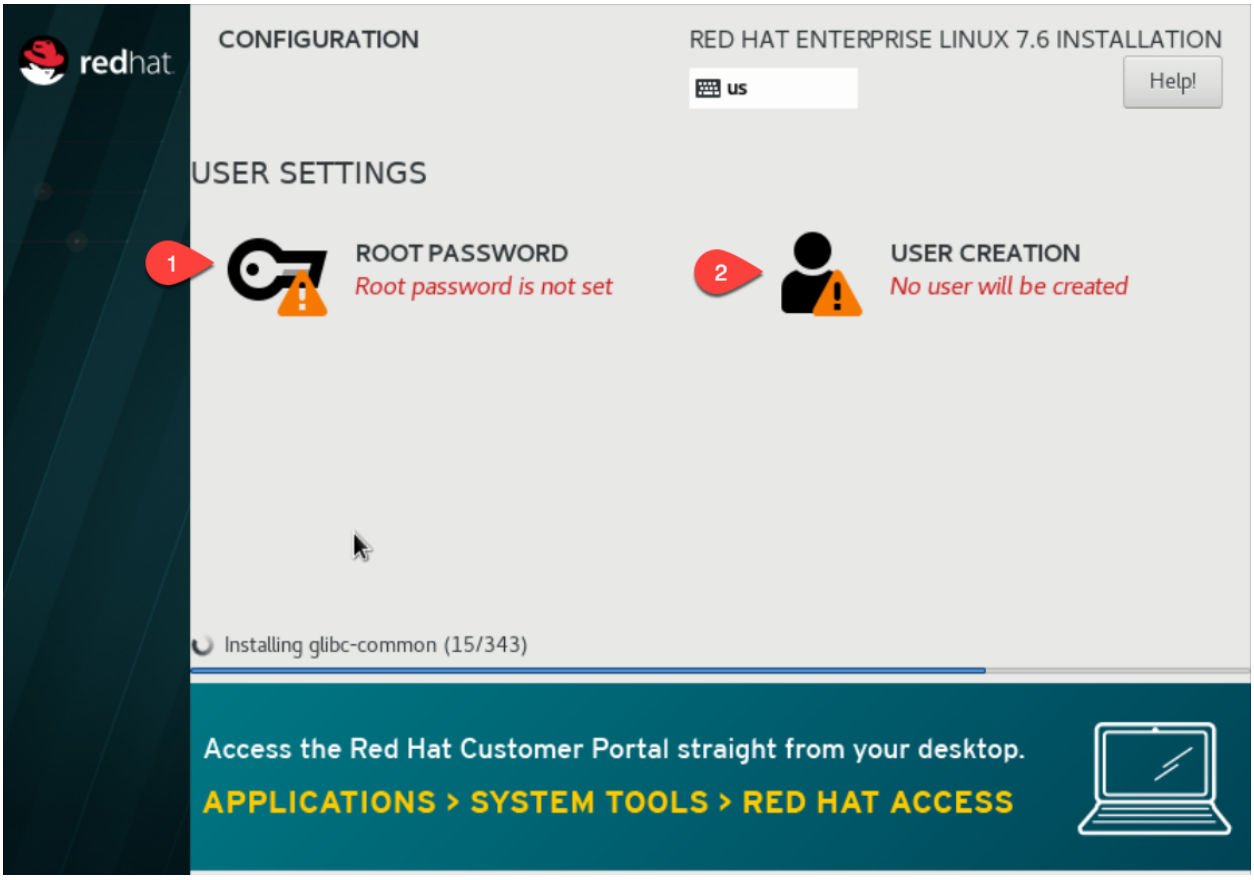


Select **DONE** review the **SUMMARY OF CHANGES**

Click **BEGIN INSTALLATION**

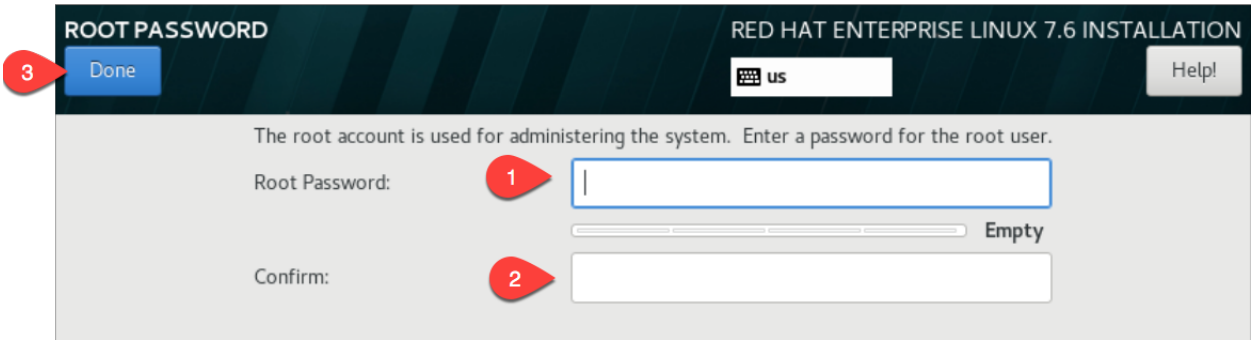
While RHEL is installing, further installation steps need to be completed.

Configure the **ROOT PASSWORD** and the **USER CREATION**

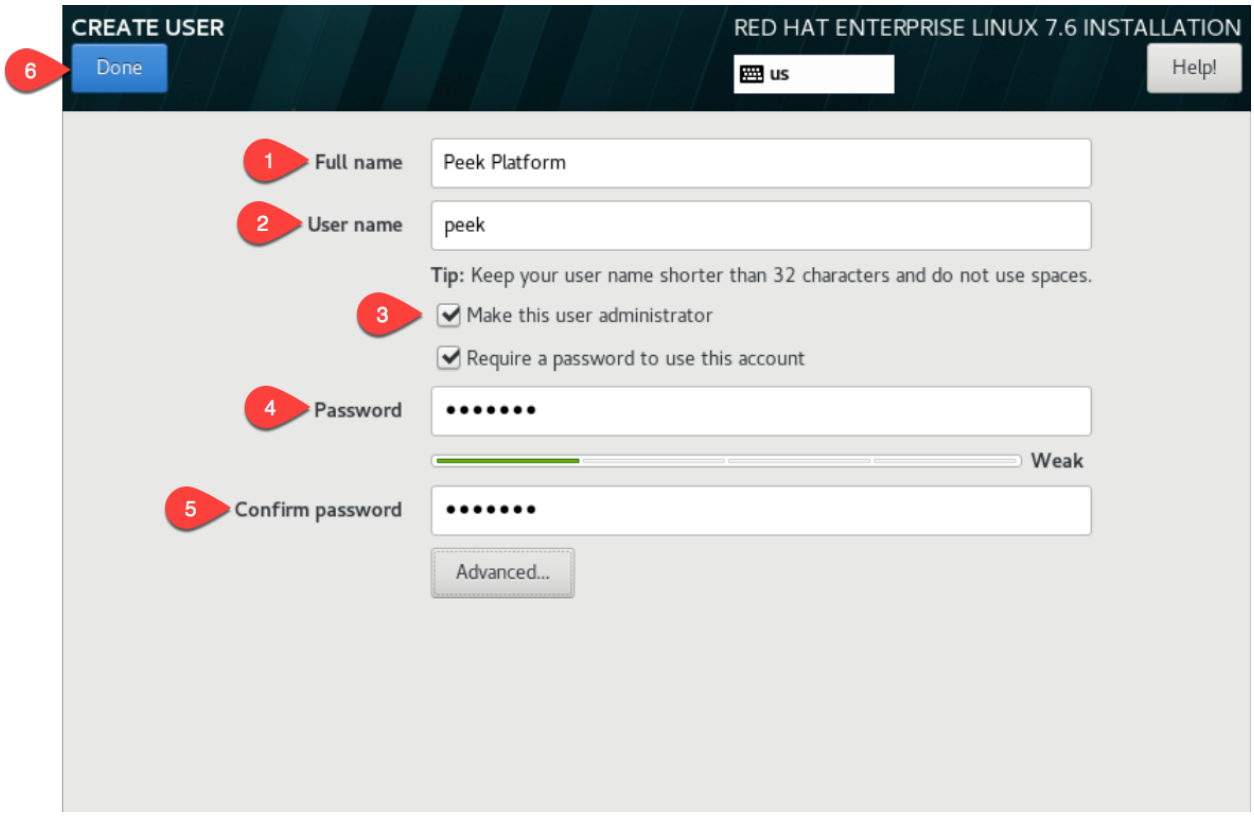


Configure the root password of the new RHEL VM.

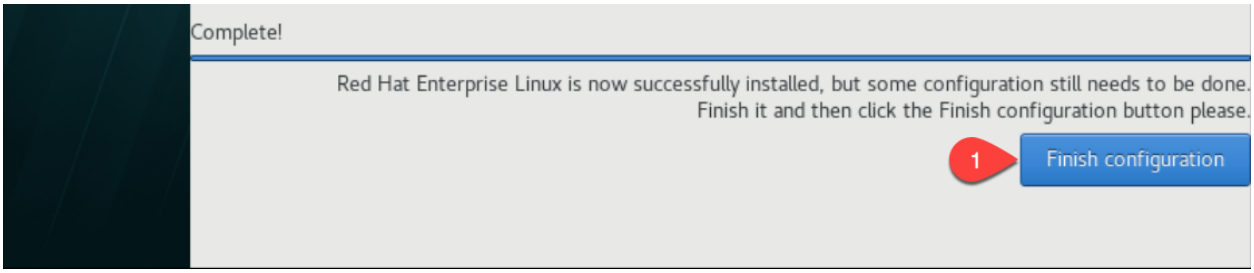
Enter the root password twice, then click **Done**



Create the **peek** user as follows.



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.

3.3.4 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.

3.3.5 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
```

3.3.6 Removing IPv6 Localhost

Run the following command to ensure that localhost does not resolve to ::1 as this effects the PostGreSQL connection.

```
sudo sed -i '/::1/d' /etc/hosts
```

3.3.7 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 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 C libraries that some python packages link to when they install:

```
# For the cryptography package
PKG="libffi-devel"

sudo yum install -y $PKG
```

Install C libraries required for LDAP:

```
PKG="openldap-devel"

sudo yum install -y $PKG
```

Install C libraries 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 C libraries 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
```

3.3.8 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.

3.3.9 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
```

3.3.10 Install PostGreSQL

Install the relational database Peek stores its data in. This is PostGreSQL 10.

Note: Run the commands in this step as the *peek* user.

Setup the PostGreSQL repository:

```
PKG="https://download.postgresql.org/pub/repos/yum/10/redhat/rhel-7-x86_64/pgdg-
↪redhat10-10-2.noarch.rpm"
sudo yum install -y $PKG
```

Install PostGreSQL:

```
PKG="postgresql10"
PKG="$PKG postgresql10-server"
PKG="$PKG postgresql10-contrib"
PKG="$PKG postgresql10-devel"
PKG="$PKG postgresql10-libs"
sudo yum install -y $PKG
```

Create the PostGreSQL cluster and configure it to auto start:

```
sudo /usr/pgsql-10/bin/postgresql-10-setup initdb
sudo systemctl enable postgresql-10
sudo systemctl start postgresql-10
```

Allow the peek OS user to login to the database as user peek with no password

```
F="/var/lib/pgsql/10/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
```

Create the peek SQL user:

```
sudo su - postgres
createuser -d -r -s peek
exit # exit postgres user
```

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
```

Create the database:

```
createdb -O peek peek
```

Cleanup traces of the password:

```
[ ! -e ~/.psql_history ] || rm ~/.psql_history
```

3.3.11 Compile and Install Python 3.6

The Peek Platform runs on Python. These instructions download, compile and install the latest version of Python.

Edit *~/.bashrc* and append the following to the end of the file.

```
##### SET THE PEEK ENVIRONMENT #####
# Setup the variables for PYTHON
export PEEK_PY_VER="3.6.8"
export PATH="/home/peek/cpython-${PEEK_PY_VER}/bin:$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.

Download and unarchive the supported version of Python:

```
cd ~
source .bashrc
wget "https://www.python.org/ftp/python/${PEEK_PY_VER}/Python-${PEEK_PY_VER}.tgz"
tar xzf Python-${PEEK_PY_VER}.tgz
```

Configure the build:

```
cd Python-${PEEK_PY_VER}
./configure --prefix=/home/peek/cpython-${PEEK_PY_VER}/ --enable-optimizations
```

Make and Make install the software:

```
make install
```

Cleanup the download and build dir:

```
cd
rm -rf Python-${PEEK_PY_VER}*
```

Symlink the python3 commands so they are the only ones picked up by path:

```
cd /home/peek/cpython-${PEEK_PY_VER}/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/cpython-${PEEK_PY_VER}/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/cpython-${PEEK_PY_VER}/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
```

3.3.12 Install Worker Dependencies

Install the parallel processing queue we use for the peek-worker 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
```

3.3.13 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.

Edit `~/.bashrc` and append the following to the file:

```
# Setup the variables for ORACLE
export LD_LIBRARY_PATH="/home/peek/oracle/instantclient_18_5:$LD_LIBRARY_PATH"
export ORACLE_HOME="/home/peek/oracle/instantclient_18_5"
```

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 **18.5.0.0.0**.

1. Download the ZIP "Basic Package" `instantclient-basic-linux.x64-18.5.0.0.0dbru.zip` from http://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html
2. Download the ZIP "SDK Package" `instantclient-sdk-linux.x64-18.5.0.0.0dbru.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.x64-18.5.0.0.0dbru.zip*
unzip instantclient-sdk-linux.x64-18.5.0.0.0dbru.zip*
```

3.3.14 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.

Edit `~/.bashrc` and append the following to the file:

```
# 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-0.95.81-1.el7.
↪x86_64.rpm"
sudo yum install -y $PKG

PKG="https://dl.fedoraproject.org/pub/epel/7/x86_64/Packages/f/freetds-devel-0.95.81-
↪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
```

3.3.15 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

3.4 Setup OS Requirements Debian

This section describes how to perform the setup for Debian Linux 9. 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 9, AKA Stretch

3.4.1 Installation Objective

This Installation Guide contains specific Debian Linux 8 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 3.6.x
- Postgres 10.4.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.

3.4.2 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.

3.4.3 Install Debian 8 OS

This section installs the Debian 8 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.

Staring 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.

3.4.4 SSH Setup

SSH is this documentations method of working with the Peek Debian VM.

SSH clients are availible 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.

3.4.5 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.

3.4.6 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
   ```

3.4.7 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"
PKG="$PKG unzip"

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 C libraries that some python packages link to when they install:

```
# For the cryptography package
PKG="libffi-dev"

sudo apt-get install -y $PKG
```

Install C libraries required for LDAP:

```
PKG="openldap-dev"

sudo apt-get install -y $PKG
```

Install C libraries 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 C libraries 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
```

3.4.8 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.

3.4.9 Install PostGreSQL

Install the relational database Peek stores its data in. This is PostGreSQL 10.

Note: Run the commands in this step as the `peek` user.

Add the latest PostGreSQL repository

```
F=/etc/apt/sources.list.d/postgresql.list
echo "deb http://apt.postgresql.org/pub/repos/apt/ stretch-pgdg main" | sudo tee $F
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key␣
↪add -
sudo apt-get update
```

Install PostGresQL

```
sudo apt-get install -y postgresql-10-postgis-2.4 postgresql-10
sudo apt-get clean
```

Allow the peek OS user to login to the database as user peek with no password

```
F=/etc/postgresql/10/main/pg_hba.conf
if ! sudo grep -q 'peek' $F; then
    echo "host  peek    peek    127.0.0.1/32    trust" | sudo tee $F -a
fi
```

Create the peek SQL user

```
sudo su - postgres
createuser -d -r -s peek
exit # Exit postgres user
```

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
```

Create the database

```
createdb -O peek peek
```

Cleanup traces of the password

```
[ ! -e ~/.psql_history ] || rm ~/.psql_history
```

3.4.10 Compile and Install Python 3.6

The Peek Platform runs on Python. These instructions download, compile and install the latest version of Python.

Edit **~/.bashrc** and insert the following after the first block comment.

Make sure these are before any lines like:

```
# If not running interactively, don't do anything
```

Insert :

```
##### SET THE PEEK ENVIRONMENT #####
# Setup the variables for PYTHON
export PEEK_PY_VER="3.6.8"
export PATH="/home/peek/cpython-${PEEK_PY_VER}/bin:$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 you get the new environment.

Download and unarchive the supported version of Python

```
cd ~
source .bashrc
wget "https://www.python.org/ftp/python/${PEEK_PY_VER}/Python-${PEEK_PY_VER}.tgz"
tar xzf Python-${PEEK_PY_VER}.tgz
```

Configure the build

```
cd Python-${PEEK_PY_VER}
./configure --prefix=/home/peek/cpython-${PEEK_PY_VER}/ --enable-optimizations
```

Make and Make install the software

```
make install
```

Cleanup the download and build dir

```
cd
rm -rf Python-${PEEK_PY_VER}*
```

Symlink the python3 commands so they are the only ones picked up by path.

```
cd /home/peek/cpython-${PEEK_PY_VER}/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/cpython-${PEEK_PY_VER}/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/cpython-${PEEK_PY_VER}/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
```

3.4.11 Install Worker Dependencies

Install the parallel processing queue we use for the peek-worker 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.conf

sudo systemctl restart redis
```

3.4.12 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.

Edit `~/.bashrc` and append the following to the file:

```
# Setup the variables for ORACLE
export LD_LIBRARY_PATH="/home/peek/oracle/instantclient_18_5:$LD_LIBRARY_PATH"
export ORACLE_HOME="/home/peek/oracle/instantclient_18_5"
```

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 **18.5.0.0.0**.

1. Download the ZIP "Basic Package" `instantclient-basic-linux.x64-18.5.0.0.0dbru.zip` from http://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html
2. Download the ZIP "SDK Package" `instantclient-sdk-linux.x64-18.5.0.0.0dbru.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.x64-18.5.0.0.0dbru.zip*
unzip instantclient-sdk-linux.x64-18.5.0.0.0dbru.zip*
```

3.4.13 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.

Edit `~/.bashrc` and insert the following after the first block comment

Make sure these are before any lines like:

```
# If not running interactively, don't do anything
```

Insert :

```
# 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 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
```

3.4.14 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

3.5 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

CHAPTER 4

Deploy Peek Platform

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*.

4.1 Windows

4.1.1 Deploy Virtual Environment

Open a PowerShell window.

Download the platform deploy script. This is the only step in this section that requires the internet.

```
$file = "deploy_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;
```

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 `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.

```
$dist = "C:\Users\peek\Downloads\peek_dist_win_#.#.#.zip"
PowerShell.exe -ExecutionPolicy Bypass -File $file $dist
```

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 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.

Thats it, Peek can now start services.

The platform is now deployed, see the admin page next.

- *Configuring Platform config.json*
- *Run Peek Manually*

4.2 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_platform_linux.sh peek@servername:/home/peek/deploy_platform_linux.sh`

```
uri="https://bitbucket.org/synerty/synerty-peek/raw/master/scripts/linux/deploy_
↪platform_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/`.

```
dist="/home/peek/Downloads/peek_dist_linux_#.#.#.tar.bz2"
bash deploy_platform_linux.sh $dist
```

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*

4.3 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_platform_macos.sh peek@servername:/Users/peek/deploy_platform_macos.sh`

```
file="deploy_platform_macos.sh"
uri="https://bitbucket.org/synerty/synerty-peek/raw/master/scripts/macos/$file"
curl -O $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/`.

```
dist="/Users/peek/Downloads/peek_dist_macos_#.#.#.tar.bz2"
bash $file $dist
```

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*

4.4 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
```

Install the **`tns`** command line tools again:

```
npm -g install nativescript
```

4.5 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

CHAPTER 5

Deploy Peek Plugins

5.1 Deploying a Production Release

This section deploys the plugins to a virtual environment, see *Deploy Peek Platform*.

For more information about plugin development and building plugin packages / releases see: *Develop Peek Plugins*

Important: Windows users must use bash.

Download the `plugin_release_dir.zip` file created in *Package Peek Plugins*

Create the release directory:

```
mkdir ~/plugin-release-dir
```

Change to release directory:

```
cd ~/plugin-release-dir
```

Unzip the contents of `plugin_release_dir.zip`:

```
unzip ~/Downloads/plugin_release_dir.zip
```

Ensure that you're in the Virtual Environment that you want your plugins deployed:

```
which python
```

Deploy the plugins:

```
pip install --no-index --find-links=. peek-plugin*
```

Read through *Administration* about updating the service `conf.json` files to include the deployed plugins.

Restart the `server` service

You have successfully deployed your peek plugins

5.2 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

CHAPTER 6

Administration

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 Server

This section sets up the config files for the **server** service.

Create following file and parent directory:

Windows `C:\Users\peek\peek-server.home\config.json`

Linux `/home/peek/peek-server.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_noop",
            "peek_plugin_etc"
        ]
    },
    "sqlalchemy": {
        "connectUrl": "postgresql://peek:PASSWORD@127.0.0.1/peek"
    }
}
```

6.1.2 Peek Client

This section sets up the config files for the **client** service.

Create following file and parent directory:

Windows `C:\Users\peek\peek-client.home\config.json`

Linux `/home/peek/peek-client.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
- Disable NativeScript preparing

```
{
    "frontend": {
        "nativescriptBuildPrepareEnabled": false
    },
    "plugin": {
        "enabled": [
            "peek_plugin_noop",
            "peek_plugin_etc"
        ]
    }
}
```

6.1.3 Peek Agent

This section sets up the config files for the **agent** service.

Create following file and parent directory:

Windows `C:\Users\peek\peek-agent.home\config.json`

Linux `/home/peek/peek-agent.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_noop",
            "peek_plugin_etc"
        ]
    }
}
```

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_server).source
```

Or Bash

```
which python
which run_peek_server
```

Confirm that the output contains the release you wish to use.

6.2.2 run_peek_server

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_server
```

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_client

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_client
```

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

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
```

6.2.5 Whats Next

Now that the platform is running, See the next section, admin_updating_plugin_settings

6.3 Logs

This document provides information on the Peek Platform logs.

The Peek Platform has several serices, each one of these services has their own log.

The logs are written to the `peek` users home directory, for example, the Peek Server services logfile is located at `/home/peek/peek-server.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 serivices `config.json` located in the services config directory for example `/home/peek/peek-server.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 recignised 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 entierly 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-server" and "peek-restarter".

If you want to restart peek, just restart "peek-server", the worker, agent and client will shutdown and be restarted.

The **peek-restarter** service automatically restarts the **worker**, **client** and **agent**. On windows, these services will stop when the **peek-server** 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\"' `
--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_gis_diagram_loader.\"DxfLoadState\"' `
--exclude-table-data 'pl_pof_gis_location_loader.\"ChunkLoadState\"' `
--exclude-table-data 'pl_pof_diagram_loader.\"PageLoadState\"' `
--exclude-table-data 'pl_docdb.\"DocDbDocument\"' `
--exclude-table-data 'pl_docdb.\"DocDbChunkQueue\"' `
--exclude-table-data 'pl_docdb.\"DocDbEncodedChunkTuple\"' `
--exclude-table-data 'pl_search.\"SearchIndex\"' `
--exclude-table-data 'pl_search.\"SearchIndexCompilerQueue\"' `
--exclude-table-data 'pl_search.\"EncodedSearchIndexChunk\"' `
--exclude-table-data 'pl_search.\"SearchObject\"' `
--exclude-table-data 'pl_search.\"SearchObjectRoute\"' `
--exclude-table-data 'pl_search.\"SearchObjectCompilerQueue\"' `
--exclude-table-data 'pl_search.\"EncodedSearchObjectChunk\"' `
--exclude-table-data 'pl_pof_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
```

CHAPTER 7

NativeScript App

7.1 Setup Nativescript Windows

This page contains a list of all system requirements needed to build and run NativeScript apps on Windows.

7.1.1 Installation Objective

This *Windows Nativescript Install Guide* contains specific Windows operating system requirements for development of synerty-peek.

Dependencies

This install procedure requires "Node.js 7+ and NPM 3+" as documented in the *Windows Requirements Install Guide* (WindowsRequirementsSetup.rst).

Required Software

Below is a list of all the required software:

- Google Chrome
- chocolatey
- Java JDK
- Android SDK
- nativescript NPM package
- Android Emulator

Optional Software

- VirtualBox
- GenyMotion (Synerty uses GenyMotion)

7.1.2 Online Installation Guide

Install google Chrome

Install google chrome

Install Chocolatey

Run the command prompt as an Administrator

Copy and paste the following script in the command prompt

```
@powershell -NoProfile -ExecutionPolicy unrestricted -Command "iex ((new-object net.
↪webclient).DownloadString('https://chocolatey.org/install.ps1'))" && SET PATH=%PATH
↪%;%ALLUSERSPROFILE%\chocolatey\bin
```

Restart the command prompt.

Java JDK

In the command prompt, run the following command

```
choco install jdk8 -y
```

Android SDK

In the command prompt, run the following command

```
choco install android-sdk -y
```

Restart the command prompt.

Install the required Android SDKs and the Local Maven repository for Support Libraries

```
echo yes | "%ANDROID_HOME%\tools\android" update sdk --filter tools,platform-tools,
↪android-23,build-tools-23.0.3,extra-android-m2repository,extra-google-m2repository,
↪extra-android-support --all --no-ui

echo yes | "%ANDROID_HOME%\tools\android" update sdk --filter tools,platform-tools,
↪android-25,build-tools-25.0.2,extra-android-m2repository,extra-google-m2repository,
↪extra-android-support --all --no-ui
```

Nativescript Package

Run the following command

```
npm i -g nativescript
```

Note: If you are developing, this step is required after every deploy.

Do you want to run the setup script?

Answer N

Restart the command prompt

Confirm Environment Variable ANDROID_HOME

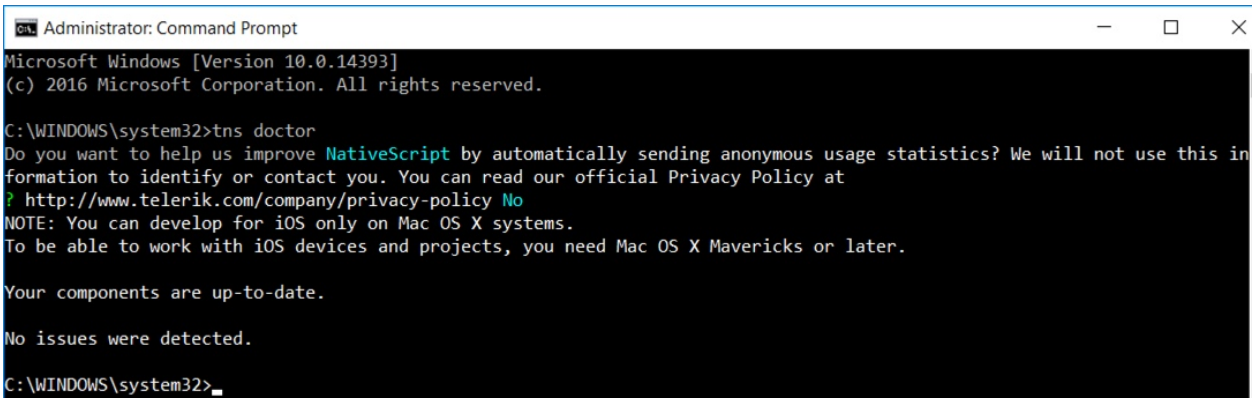
```
C:\Users\peek\AppData\Local\Android\android-sdk
```

Confirm Environment Variable JAVA_HOME

```
C:\Program Files\Java\jdk1.8.0_121
```

Check the installation with tns

```
tns doctor
```



Android Emulator Setup

You can use any emulator. Synerty has written instructions for GenyMotion.

Download - with Virtualbox https://www.genymotion.com/download/

Install GenyMotion with Virtualbox, all default options

Run both GenyMotion and Virtualbox

In GenyMotion select the add button to create a virtual device

---

Select a device and select next

Update the "Virtual device name" to something shorter (easier to remember and type) and select next

Your virtual device will be retrieved and deployed

With a device selected in the "Your virtual devices" list select the "Start" button

Your device emulation will start in a new window

7.1.3 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

7.2 Setup Nativescript Debian

TODO This is still the windows guide.

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.

7.2.1 Installation Objective

This *Installation Guide* contains specific Windows operating system requirements for the configuring of synerty-peek.

Dependencies

This install procedure requires software installed by the prerequisites steps.

Optional Software

- VirtualBox
- Geny Motion

Required Software

Below is a list of all the required software:

- Java JDK
- nativescript NPM package

7.2.2 Installation Guide

Install the 32bit requirements

```
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install ia32-libs lib32ncurses5 lib32stdc++6 lib32z1
sudo apt-get build-essential
```

Install Java JDK

```
sudo apt-get install openjdk-8-jdk
sudo update-alternatives --config java
```

Installing the SDK

Download the SDK only from here, scroll right to the bottom, to the "Get just the command line tools" section, and download the linux tools.

https://developer.android.com/studio/index.html#downloads

```
mkdir ~/android && cd ~/android
unzip /media/psf/Downloads/tools_r25.2.3-linux.zip

echo "export JAVA_HOME=$(update-alternatives --query javac | sed -n -e 's/Best: *\(.
↪*\)\/bin\/javac/\1/p')" >> ~/.bashrc
echo "export ANDROID_HOME=~/android" >> ~/.bashrc
echo "alias android=$ANDROID_HOME/tools/android" >> ~/.bashrc
echo "alias emulator=$ANDROID_HOME/tools/emulator" >> ~/.bashrc
```

Create the Android Virtual Device

```
$ANDROID_HOME/tools/android avd
```

1. Tools -> SDK Manager
2. Install packages as per .. image:: ./install_images.png
3. Create Device as per .. image:: ./create_device.png

Nativescript Package

This section installs the following:

- Nativescript command line utility (tns)
- Nativescript build tools
- Android emulator (with no images)
- Android SDK (With no SDKs)

Install the required NPM packages

Run the Command Prompt as Administrator and run the following commands:

```
npm -g install nativescript
```

Do you want to run the setup script?

```
Y
```

###.. image:: Nativescript-Install.jpg

Allow the script to install Chocolatey (It's mandatory for the rest of the script)

Answer A

Do you want to install the Android emulator?

Answer Y

Do you want to install HAXM (Hardware accelerated Android emulator)?:

Answer Y

When the blue power shell windows says it's finished, close it.

Return focus to the original window, you should see

> If you are using bash or zsh, you can enable command-line completion. > Do you want to enable it now? (Y/n)

Press "n", then "Enter".

When the script has finished: log off windows.

Login to windows as peek, Then open a command window and continue.

—

Check the installation with tns

```
tns doctor
```

Note: At this point you may find your self in a real life infinite loop. as tns doctor may ask you to run the setup script again if the setup is broken.

Confirm Environment Variable ANDROID_HOME

```
C:\Users\peek\AppData\Local\Android\android-sdk
```

Confirm Environment Variable JAVA_HOME

```
C:\Program Files\Java\jdk1.8.0_121
```

Note: For Offline installation, install the Node.js 7+ and NPM 3+ on a machine with internet access. Package the installed nodejs files and installed modules 'C:Userspeeknodejs'. Unpackage in the same directory location on the offline server.

7.2.3 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

7.3 Setup Nativescript MacOS

The Peek platform is designed to run on Linux, however, it is compatible with macOS. Please read through all of the documentation before commencing the installation procedure.

7.3.1 Installation Objective

This *Installation Guide* contains specific macOS operating system requirements for the configuring of synerty-peek.

Dependencies

This install procedure requires software installed by the prerequisites steps.

Optional Software

- Android Studio
- VirtualBox
- Geny Motion

Required Software

Below is a list of all the required software:

- nativescript NPM package

7.3.2 Installation Guide

Dependencies for iOS development

Install the xcodeproj ruby gem with the following command.

```
sudo gem install xcodeproj
```

Install CocoaPods

```
sudo gem install cocoapods
```

Dependencies for Android development

Android Studio

Android Studio is required if you intend to develop the Android NativeScript app.

Download Android Studio

Launch the Android Studio DMG file.

Drag and drop Android Studio into the Applications folder, then launch Android Studio.

Select whether you want to import previous Android Studio settings, then click OK.

The Android Studio Setup Wizard guides you though the rest of the setup, which includes downloading Android SDK components that are required for development.

Installation Type

Select Custom

If you want to run the SDK tools virtual emulator, check the following, or leave them unchecked if you want to use Geny Motion.

Check Performance

Check Android Virtual Device

Note: If you're install inside a virtual machine you'll get the following message during the installation of Android Studio.

Unable to install Intel HAXM HAXM doesn't support nested virtual machines. Unfortunately, the Android Emulator can't support virtual machine acceleration from within a virtual machine.

SDK Manager

In the Android Manager Welcome screen open the `Configure` drop down at the bottom of the window and select `SDK Manager`

Go to the `SDK Platforms` tab

At the bottom of the window:

Check `Show Package Details`

In the list:

Check Android 7.1.1 `Android SDK Platform 25`

Uncheck Unselect the other APIs

Go to the `SDK Tools` tab

At the bottom of the window:

Check `Show Package Details`

In the list:

Check Android SDK Build-Tools `25.0.3`

Uncheck Unselect the other versions.

Select 'ok' and confirm the install

Close Android Studio

7.3.3 Nativescript Package

Install the required NPM packages

Create android dummy repositories file:

```
touch ~/.android/repositories.cfg
```

Create symlinks for NativeScript install:

```
ln -s /Users/peek/Library/Android/sdk /usr/local/opt/android-sdk

# Find the version of java that you have:
ls -d /Library/Java/JavaVirtualMachines/jdk1.8.0_*

# Set the version of java or just leave this as * if there is only one.
sudo ln -s /Library/Java/JavaVirtualMachines/jdk1.8.0_*.jdk/Contents/Home /Library/
↪Java/Home
```

Edit `~/.bash_profile` and insert the following after the first block comment.

Make sure these are before any lines like:

```
# If not running interactively, don't do anything
```

Insert :

```
##### SET THE ANDROID ENVIRONMENT #####
export ANDROID_HOME="/Users/peek/Library/Android/sdk"
```

Warning: Close and reopen the terminal to ensure the profile takes effect.

Run the following command in a new terminal:

```
npm -g install nativescript@latest typescript tslint node-sass
```

Do you want to run the setup script?

Answer Y

Do you have Xcode installed (Xcode was installed during the OS Requirements Setup)?

Answer Y

---

software license agreements:

Answer Type `q`, `agree` and hit 'enter'

Allow the script to install Homebrew?

Answer N

Allow the script to install Java SE Development Kit?

Answer N

Allow the script to install Android SDK?

Answer N

Allow the script to install CocoaPods?

Answer Y

Allow the script to install xcodeproj?

Answer Y

Do you want to install Android emulator?

Answer N

Check the installation with `tns` in a new terminal:

```
tns doctor
```

Note: At this point you may find your self in a real life infinite loop. as tns doctor may ask you to run the setup script again if the setup is broken.

7.3.4 Android Emulator Setup

You can use any emulator. Synerty has written instructions for GenyMotion.

Warning: If you've setup your development console in a VM, you'll need to install the Android emulator on the host machine. Skip to these instructions: *Android Emulator Setup for VM*.

Download and Install VirtualBox

Download http://download.virtualbox.org/virtualbox/5.1.26/VirtualBox-5.1.26-117224-OSX.dmg

---

Install GenyMotion, all default options

Download https://www.genymotion.com/download/

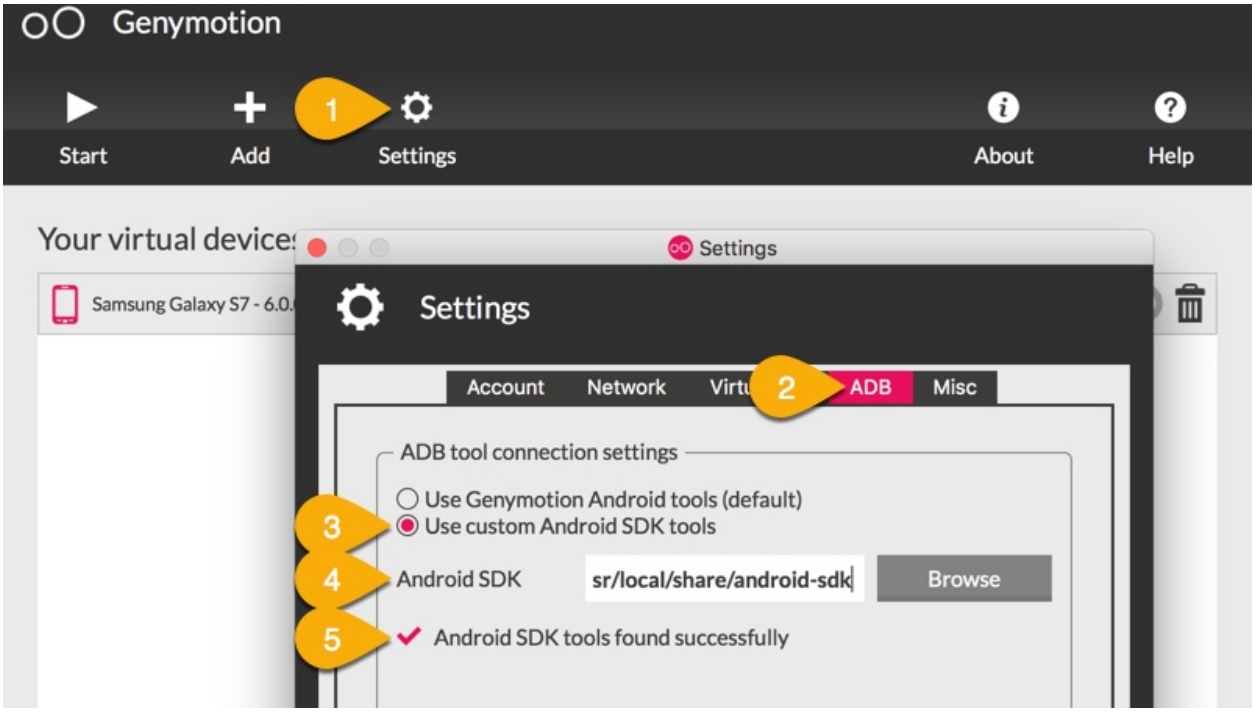
Run GenyMotion

Create Android device

1. Select the 'Add' button to create a virtual device
2. Select a device and select next
3. Update the "Virtual device name" to something shorter (easier to remember and type) and select next

Your virtual device will be retrieved and deployed

ABD Tool Connection Settings



1. Select 'Settings'
2. Select the 'ABD' tab
3. Check the 'Use custom Android SDK tools'
4. Paste `/Users/peek/Library/Android/sdk`
5. Confirm the the Android SDK tools are found successfully

With a device selected in the "Your virtual devices" list select the "Start" button

Your device emulation will start in a new window

In a terminal run `tns device` to check tns can find your device.

7.3.5 Android Emulator Setup for VM

If you've setup your development console in a VM, you'll need to install the Android emulator on the **HOST MACHINE**.

Follow the *Android Emulator Setup* instructions on the host machine then continue the following these instructions.

> **Warning:** If you are **NOT** using a VM these instructions are not required.

Go to the **HOST MACHINE**.

With your emulator device started, run the following commands in terminal:

```
adb shell ifconfig
adb tcpip 5556
```

Go to the **VM** and run the following commands in terminal.

Install Android Platform Tools:

```
brew cask install android-platform-tools
```

Connect to your genyMotion device:

```
adb connect <ip_of_genymotion>:5556
```

List attached devices:

```
adb devices
```

Change to the `build_ns` directory, check that tns can find the device:

```
tns devices
```

7.3.6 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

7.4 Package NativeScript App

Note: The Windows or Debian setup NativeScript must be followed before following this guide.

To deploy the NativeScript App, you may use a Synerty provided release or build your own.

A release is a zip file containing all the required node_modules.

7.4.1 Windows

This section contains the steps to build your own NativeScript App release.

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 package NativeScript App dependencies script. Run the following commands in the PowerShell window.

```
$file = "package_nativescript_app_win.ps1";
$uri = "https://bitbucket.org/synerty/synerty-peek/raw/master/$file";
Invoke-WebRequest -Uri $uri -UseBasicParsing -OutFile $file;
```

Run the package NativeScript App dependencies script.

```
PowerShell.exe -ExecutionPolicy Bypass -File package_nativescript_app_win.ps1
```

The script will download the NativeScript App dependencies.

Take note of the end of the script, it will print out where the release is.

7.4.2 Linux

TODO

7.4.3 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

7.5 Deploy NativeScript App

Note: The Windows or Debian setup NativeScript must be followed before following this guide.

This section describes how to deploy a NativeScript App.

Peek is deployed into python virtual environments, a new virtual environment is created for every deployment.

To package your own NativeScript App dependencies, see the following document *Package NativeScript App*.

7.5.1 Windows

Open a PowerShell window.

Download the NativeScript App dependencies deploy script. This is the only step in this section that requires the internet.

```
$file = "deploy_nativescript_app_win.ps1"
$uri = "https://bitbucket.org/synerty/synerty-peek/raw/master/$file";
Invoke-WebRequest -Uri $uri -UseBasicParsing -OutFile $file;
```

Run the deploy NativeScript App dependencies script. The script will complete with a print out of the environment the NativeScript App dependencies were deployed. Ensure you update the **$ver** variable with the environment version you're deploying. Also update the **$dist** variable with the path to your release.

The script will deploy to `C:\Users\peek`.

```
$ver = "#.#.#"
$dist = "C:\Users\peek\Downloads\peek_dist_nativescript_app_win.zip"
PowerShell.exe -ExecutionPolicy Bypass -File deploy_nativescript_app_win.ps1 $ver
↪$dist
```

The NativeScript App dependencies are now deployed.

7.5.2 Linux

TODO

7.5.3 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

7.6 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

CHAPTER 8

Peek Development

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 familar with the following:

- GIT
- Python3.5+
- Python Twisted
- HTML
- CSS
- Bootstrap3
- TypeScript
- Angular (Angular2+, not AngularJS aka Angular1)
- NativeScript

Note: This a cross platform development guide, all commands are writen 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.

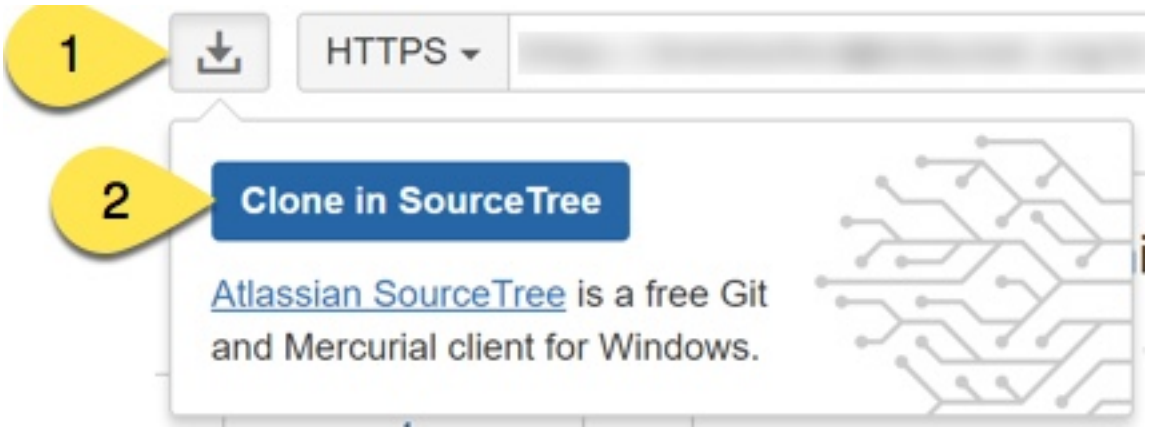
Clone a New Peek Plugin

If you're creating a new plugin you can copy from "peek-plugin-noop" and rename.

Copy peek-plugin-noop

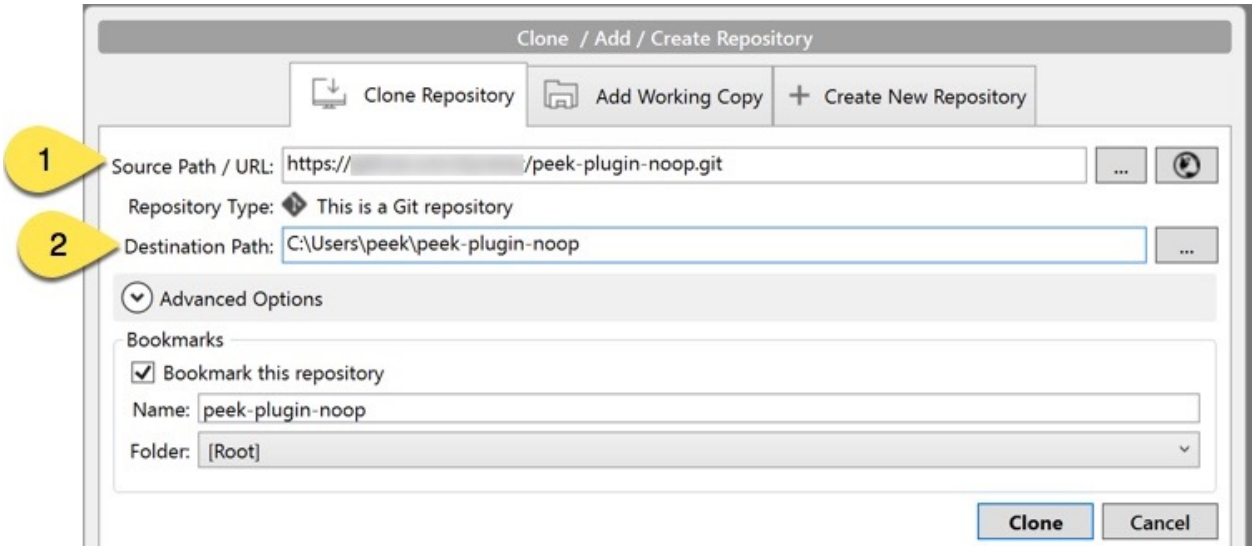
Clone https://bitbucket.org/synerty/peek-plugin-noop/src

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:

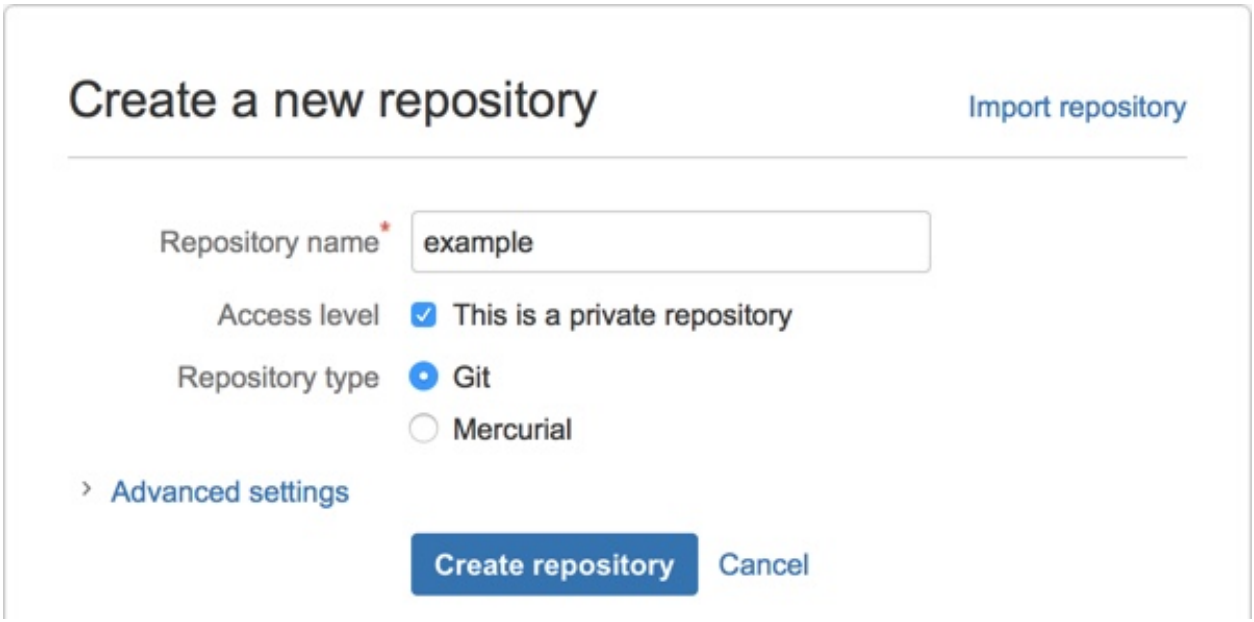
```
bash ./rename_plugin.sh
```

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.



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}
```

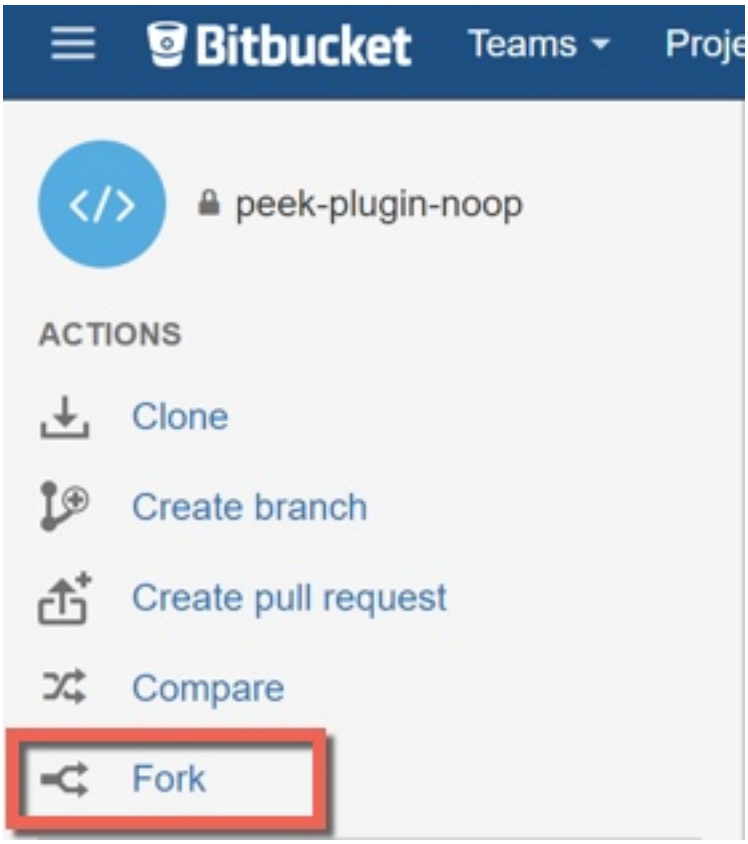
Push your changes:

```
git push -u origin master
```

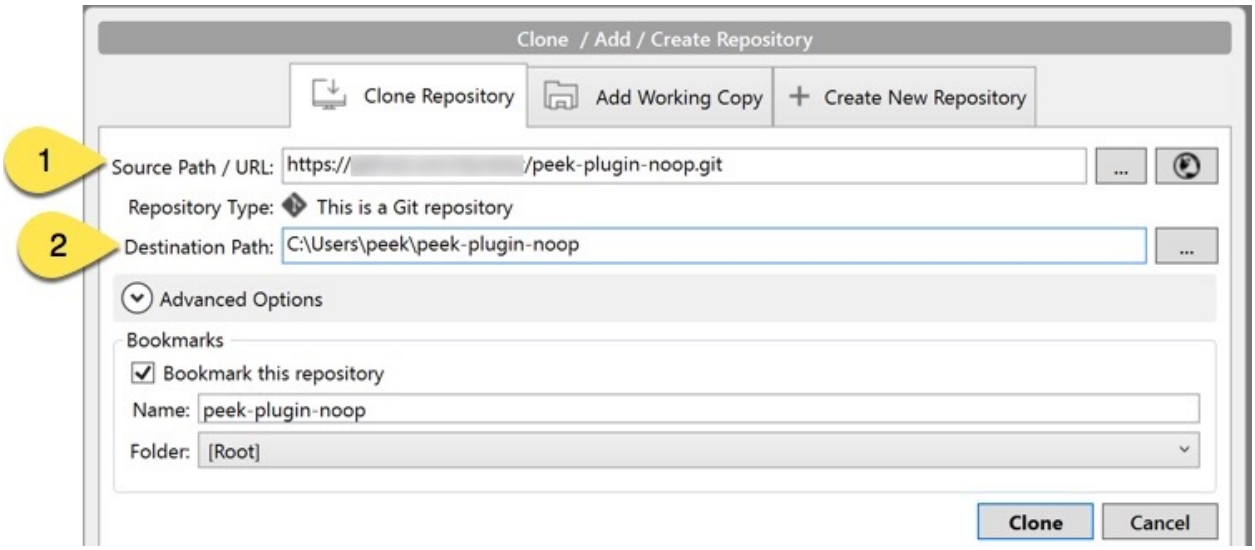
Cloning an Existing Peek Plugin

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`** in the *Deploy Peek Plugins*.

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**, **client** and **server** 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_server`** 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 server
run_peek_server

# Run the peek client
run_peek_client

# Run the peek agent
run_peek_agent

# Run the peek worker
run_peek_worker
```

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
- Mobile builds:
  - A NativeScript app, using NativeScript
  - A Web app, using @angular/cli
- Desktop 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/frontend/

Combining Files

The Server and Client 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 Client and Server services. There were many things to consider when implementing this code, consideration 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. Instead of using Trickery with `tsconfig.json` and special NPM packages that switch between NativeScript and Web dependencies, Peek rewrites parts of the frontend code on the fly. For example this method rewrites Web files to work with the NativeScript frontend. Here is an example, NativescriptBuilder.py.

Angular Ahead of Time Compilation. The final nail in the symlink approach was Angular AoT Compilation support. With the new file copy approach, Peek does away with on the fly switching of NativeScript VS Angular dependencies.

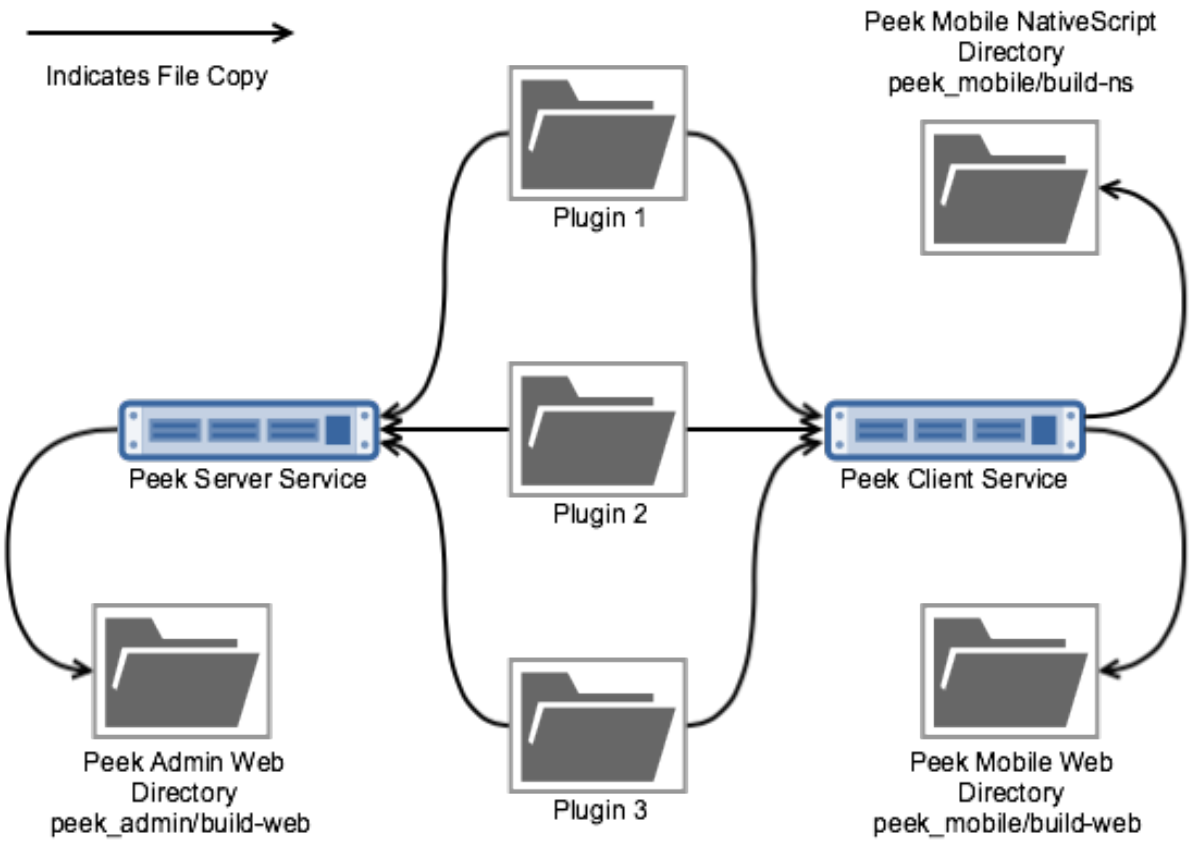
Plugins in node_modules. Plugins in the frontends can leverage each other. For example, other plugins can leverage services from `peek_core_user` for user authentication. To do this, Peek copies files into node_modules. However, nativescript doesn't recompile these files during live sync, so Peek also compiles them and copies them to the platforms directory.

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 for NativeScript
6. Copy required files to the platform directory for NativeScript.

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 build-web, for NativeScript use the `~/peek-server.home/frontendNodeModuleOverlayDir` or `~/peek-client.home/frontendNodeModuleOverlayDir`.

Peek reads files from either the `~/peek-server.home/frontendSrcOverlayDir` or `~/peek-client.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 NativeScript 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 Server and Peek Client `config.json` files.

```
{
    ...
    "frontend": {
        ...
        "frontendSrcOverlayDir": "/home/peek/peek-client.home/frontendSrcOverlayDir",
```

(continues on next page)

(continued from previous page)

```
        "frontendNodeModuleOverlayDir": "/home/peek/peek-client.home/
↪frontendNodeModuleOverlayDir",
    },
    ...
}
```

#1 Copy the plugin build directory into the frontend folder

```
cp -r synerty-peek-1.3.4/lib/python3.6/site-packages/peek_desktop/build-web/src/peek_
↪plugin_pof_diagram peek-client.home/frontendSrcOverlayDir/
```

#2 Remove the files not being updated (we're updating pofDiagram.component.web.html)

```
cd peek-client.home/frontendSrcOverlayDir/peek_plugin_pof_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_pof_diagram
```

5 Undo the changes

```
rm -rf ~/peek-client.home/frontendSrcOverlayDir/peek_plugin_pof_diagram
restart_peek.sh
```

Live Updating for Development

Both **NativeScript** 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, NativeScript will compile the TypeScript, redeploy the javascript to the native app and reload the NativeScript.

Peeks 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-server.home/config.json` or `~/peek-client.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 Server and Client restarts. Set `frontend.webBuildEnabled` to `false`.

If DEBUG logging is also enabled, you'll see Peek working away when you change files.

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

(continues on next page)

(continued from previous page)

```
        "syncFilesForDebugEnabled": true,
        "webBuildEnabled": false,
        ....
    },
    "logging": {
        "level": "DEBUG"
    },
    ...
}
```

Now when you run:

```
# Start Angular CLI live dev server
npm start
```

Or

```
# Start NativeScript live sync
tns run <Platform>
```

The NativeScript and 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.

build-ns

Running the command `tns device` will list active virtual devices and connected physical devices

```
$ tns device

Connected devices & emulators
Searching for devices...
─────────────────────────────────────────────────────────────────────────────────────────────────────────────────
│ # │ Device Name          │ Platform │ Device Identifier                          │␣
↪Type      │ Status     │
│ 1 │ Synerty 008 iPad     │ iOS      │ a8f83ceb9ddd5d0df25d618a5a4c6d9bf7a6f5f9 │␣
↪Device    │ Connected │
│ 2 │ iPad Pro (9.7 inch) │ iOS      │ 57AF4696-FB0A-4E42-94EB-37C38164AAB6     │␣
↪Emulator │ Connected │
└────────────────────────────────────────────────────────────────────────────────────────────────────────────────
```

tns development build command builds the project for the selected target platform and produces an application package or an emulator package:

```
tns build <Platform>
```

tns development run command runs your project on a connected device or in the native emulator, if configured:

```
       tns run <Platform>


or: ::

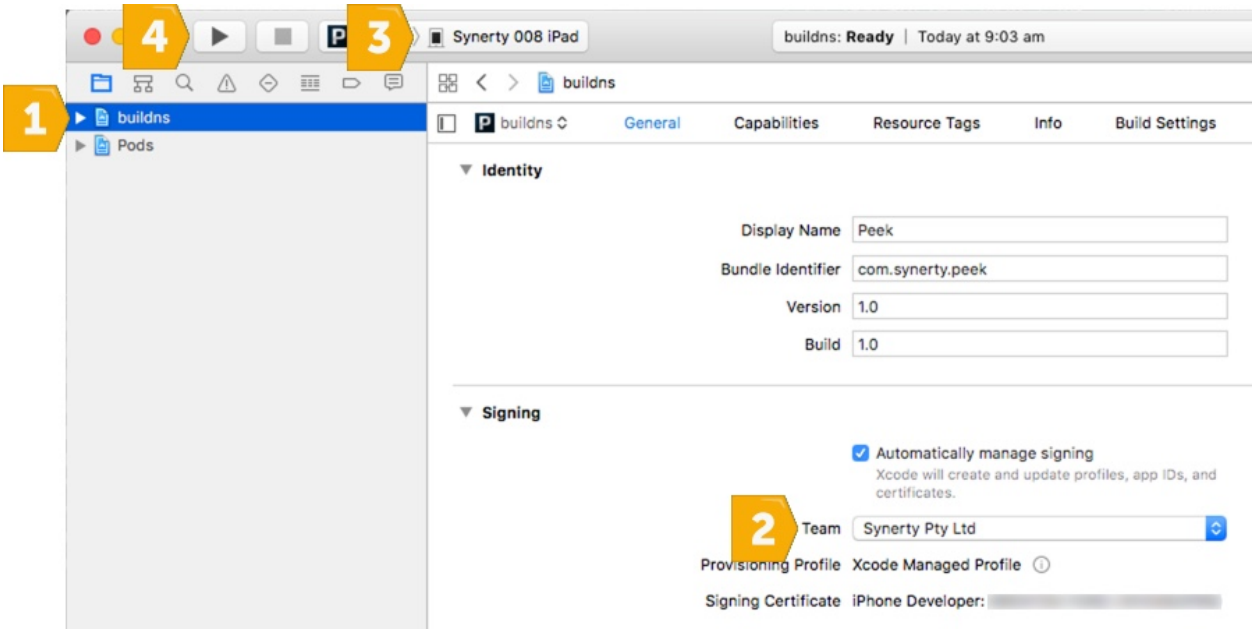
       tns run <Device ID>
```

Developing on iOS Devices

Before Peek can be deployed the signing certificate must be transfered to the device using Xcode.

To develop with iOS you'll need a developer account on https://developer.apple.com

Build the iOS Platform directory `tns build ios` then open the `build-ns/platform/ios` directory with Xcode.



1. Select the `buildns` project
2. Select the Apple Developer Team
3. Select the connected physical device
4. Deploy Peek to the device

After following this procedure you can then use `tns` to deploy Peek as the certificate will remain on the device.

Troubleshooting

OSError: inotify instance limit reached

If you receive an error when starting the server or client 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 *Learn Plugin Development*.

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

...

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 uncomitted changes
- Updates version numbers on variose 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 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 familar with the following:

- GIT
- Python3.5+
- Python Twisted
- HTML
- CSS
- Bootstrap3
- TypeScript
- Angular (Angular2+, not AngularJS aka Angular1)
- NativeScript

Note: This a cross platform development guide, all commands are writen 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 https://bitbucket.org/account/user/synerty/projects/PEEK

- synerty-peek
- peek-plugin-base

- peek-agent
- peek-client
- peek-mobile
- peek-platform
- peek-server
- peek-admin
- peek-worker

Clone Peek Repositories

Checkout repositories all in the same folder

https://bitbucket.org/synerty/synerty-peek.git

Use this script to insert individual peek modules. Update {gitAccount} and {repository} in the script below:

```
REPO="{repository}"

if [ ! -d ~peek/peek-dev ]; then
    mkdir ~peek/peek-dev
    cd ~peek/peek-dev/
    git clone https://bitbucket.org/synerty/$REPO.git
    cd ~peek/peek-mobile/$REPO
    git config --unset core.symlink
    git config --add core.symlink true
else
    echo "ALERT: `pwd` directory already exists.  Please investigate then retry."
fi

cd ~peek/peek-dev/
ls -l
```

Use this script to clone all repositories. Update {gitAccount} in the script below:

```
REPOS="synerty-peek"
REPOS="$REPOS peek-plugin-base"
REPOS="$REPOS peek-agent"
REPOS="$REPOS peek-client"
REPOS="$REPOS peek-mobile"
REPOS="$REPOS peek-platform"
REPOS="$REPOS peek-server"
REPOS="$REPOS peek-admin"
REPOS="$REPOS peek-worker"

if [ ! -d ~peek/peek-dev ]; then
mkdir ~peek/peek-dev
cd ~peek/peek-dev/
for REPO in ${REPOS[*]}
do
    echo $REPO
    git clone https://bitbucket.org/synerty/$REPO.git
    cd ~peek/peek-dev/$REPO
    git config --unset core.symlink
    git config --add core.symlink true
```

(continues on next page)

(continued from previous page)

```
    cd ~peek/peek-dev/
done
else
    cd ~peek/peek-dev/
    echo "ALERT: `pwd` directory already exists.  Please investigate then retry."
fi
ls -l
```

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.

Install Front End Modules

Remove the old npm modules files and re-install for both client and server front and packages. Run the following commands:

```
cd ~peek/peek-dev/peek-mobile/peek_mobile/build-web
[ -d node_modules ] && rm -rf node_modules
npm install
cd ~peek/peek-dev/peek-mobile/peek_mobile/build-ns
[ -d node_modules ] && rm -rf node_modules
npm install
cd ~peek/peek-dev/peek-admin/peek_admin/build-web
[ -d node_modules ] && rm -rf node_modules
npm install
```

Install synerty-peek Dependencies

These steps link the projects under site-packages and installs their dependencies.

For synerty-peek, run the following commands:

```
cd ~peek/peek-dev/synerty-peek
./pip_uninstall_and_develop.sh
```

For repositories and plugins, run from their directory

```
python setup.py develop
```

Compile Front End Packages

Symlink the tsconfig.json and node_modules file and directory in the parent directory of peek-mobile, peek-admin and the plugins. These steps are run in the directory where the projects are checked out from. These are required for the frontend typescript compiler.

Run the following commands:

```
cd ~peek/peek-dev/peek-mobile/peek_mobile/build-web
ng build
cd ~peek/peek-dev/peek-admin/peek_admin/build-web
ng build
```

Develop

You are ready to develop synerty-peek services

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/peek-dev/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/peek-dev/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.

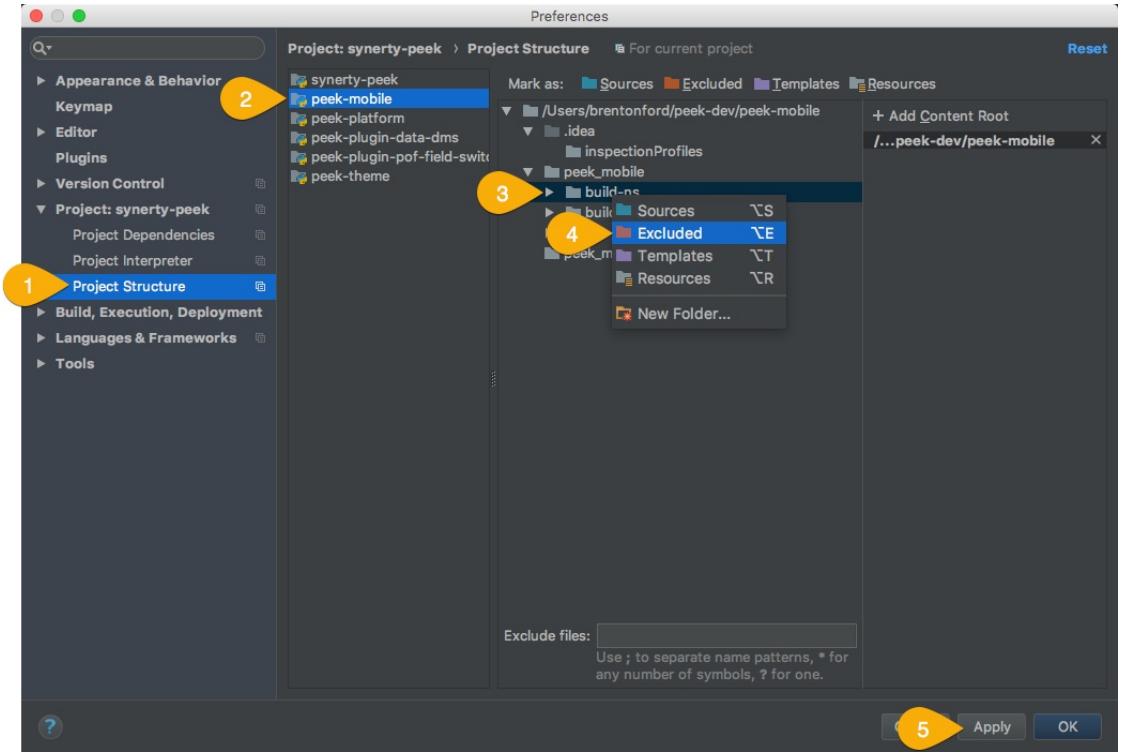
Take note of the end of the script, it will print out where the release is.

8.6.4 What Next?

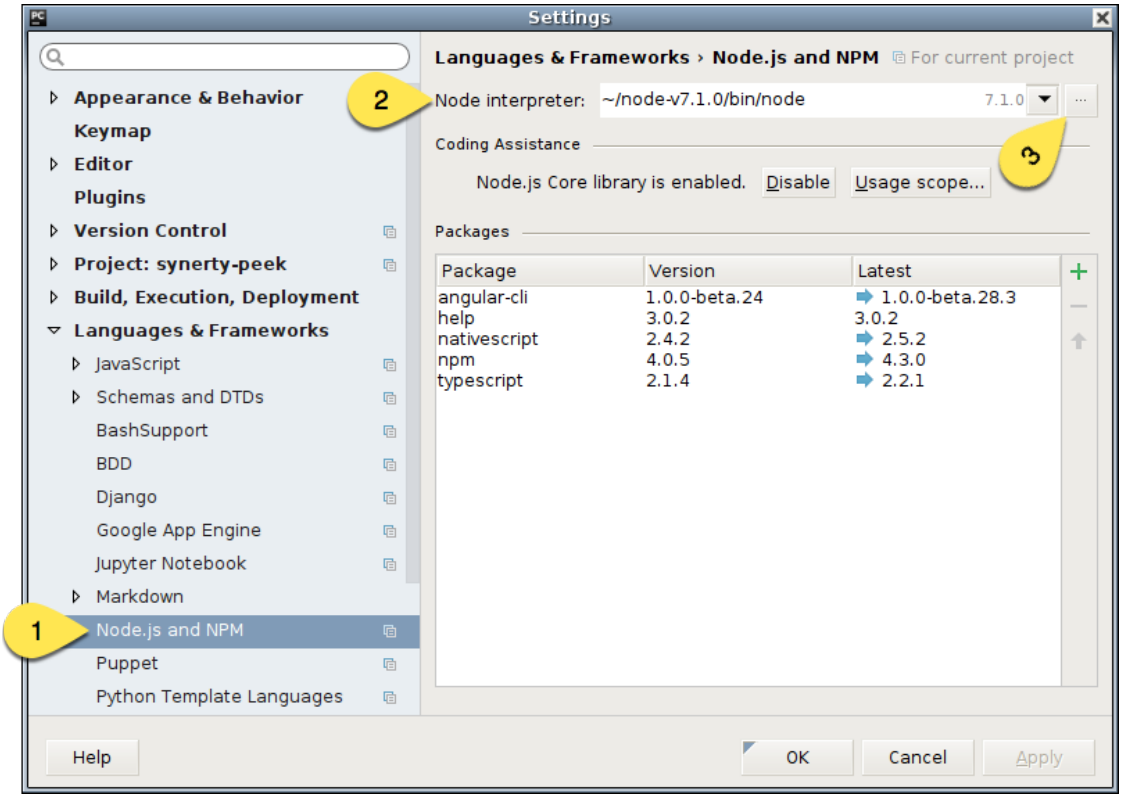
Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

8.7 Setup Pycharm IDE

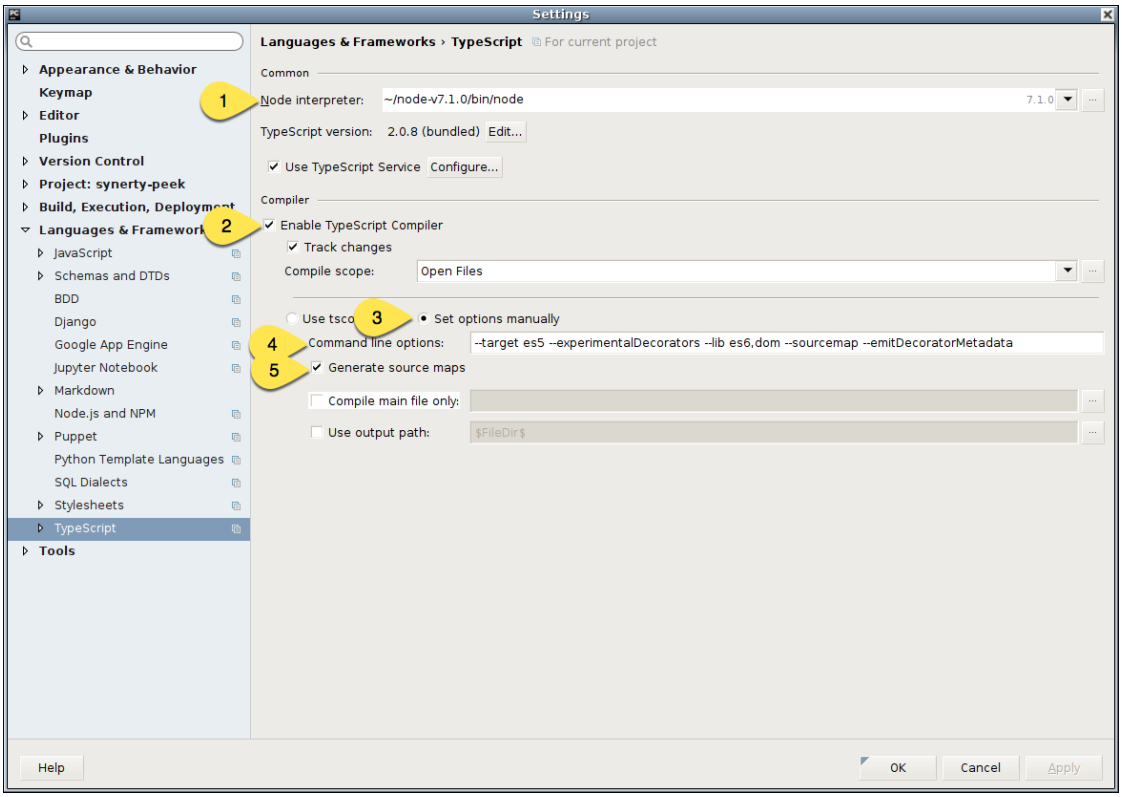
1. Open pycharm,
    1. Open the peek project, open in new window
    2. Open each of the other projects mentioned above, add to current window
2. File -> Settings (Ctrl+Alt+S with eclipse keymap)
    1. Editor -> Inspection (use the search bar for finding the inspections)
        1. Disable Python -> "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_server depends on -> peek_platform, peek_admin
        3. peek_client depends on -> peek_platform, peek_mobile
        4. peek_agent depends on -> peek_platform
        5. peek_worker depends on -> peek_platform
    3. Project -> Project Structure
        1. peek-mobile -> build-ns -> Excluded (as per the image below)
        2. peek-desktop -> build-web -> Excluded
        3. peek-admin -> build-web -> Excluded



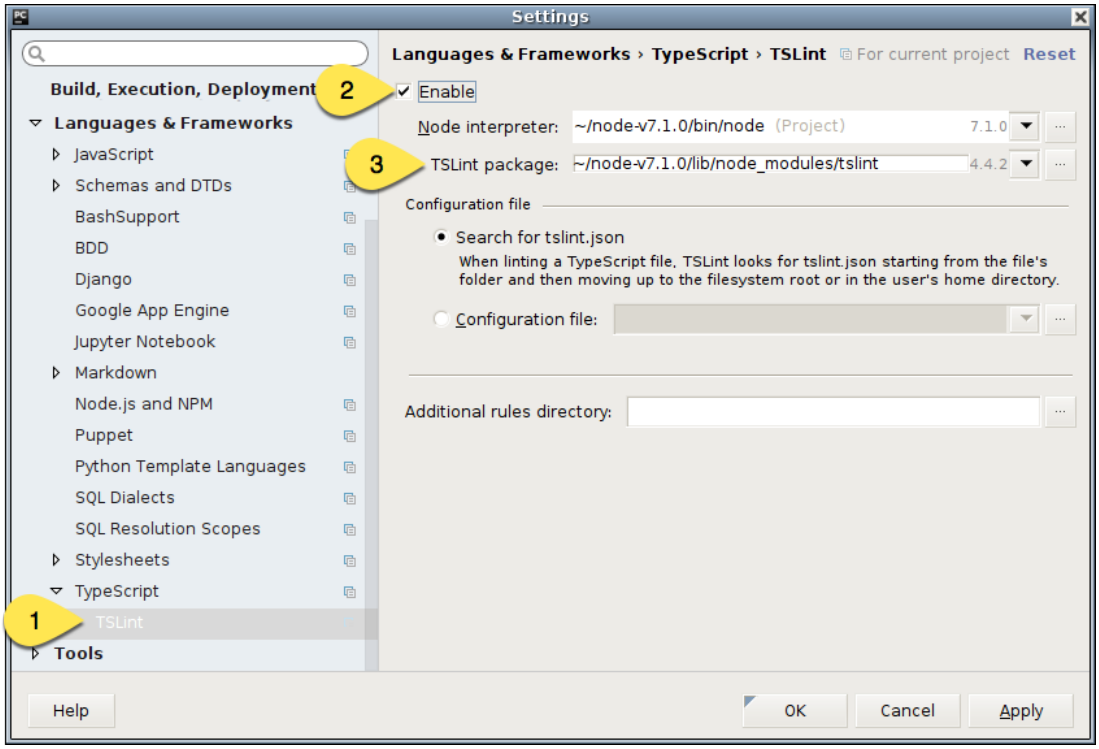
4. Languages & Frameworks -> Node.js and NPM
    1. Node interpreter -> ~/node-v7.1.0/bin/node
    2. Remove other node interpreters



5. Languages & Frameworks -> TypesScript
    1. Node interpreter -> ~/node-v7.1.0/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

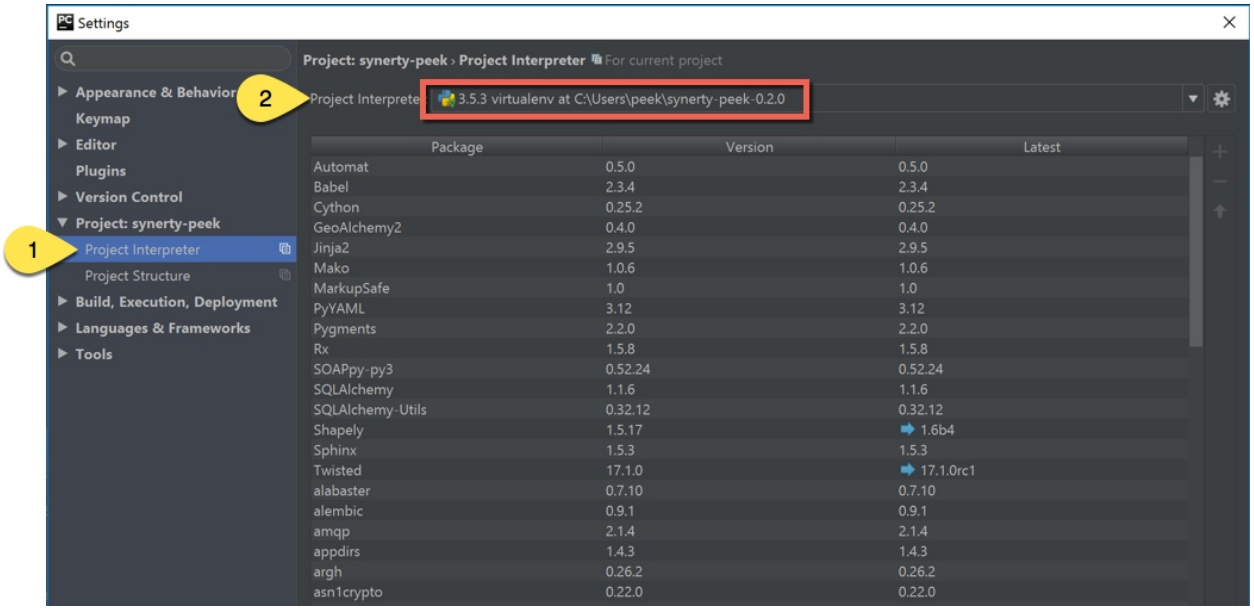


6. Languages & Frameworks -> Typescript -> TSLint
    1. Select “Enable”
    2. Node interpreter -> ~/node-v7.1.0/bin/node
    3. TSLint Package -> ~/node-v7.1.0/lib/node_modules/tslint



Configure your developing software to use the virtual environment you wish to use

Here is an example of the setting in PyCharm:



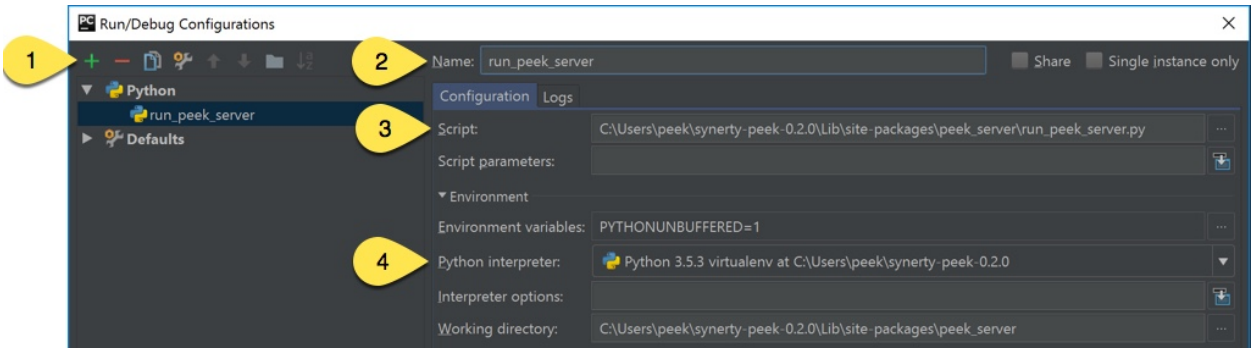
Restart the services that use the plugin

Note: The plugins that aren't being developed should be installed as per *Deploy Peek Plugins*

This is an example of running the server 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 Learn Plugin Development

8.9.1 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 *peek_plugin_base*.

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.

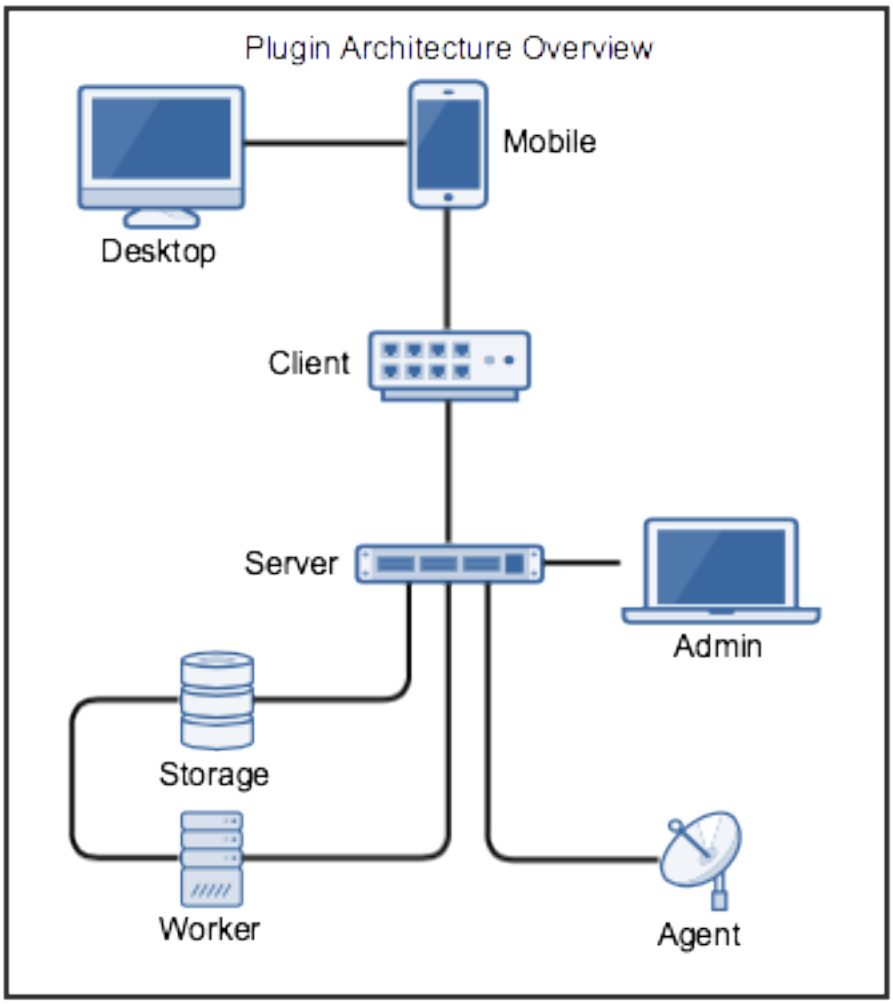
```
which python
```

This should return the location of your virtual environment, usually `~/synerty-peek-V.E.R/bin/python` on Linux or 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 :



8.9.2 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 hypons instead of underscores, Python can't import directories with hypons, 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
.idea
.vscode
docs/api_autoapi
```

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__ = '0.0.1'" > 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 = '0.0.1'
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=""
```

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, desktop and mobile)
- 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": "0.0.11",
        "buildNumber": "#PLUGIN_BUILD#",
        "buildDate": "#BUILD_DATE#",
        "creator": "Synerty Pty Ltd",
        "website": "www.synerty.com"
    },
    "requiresServices": [
    ],
    "admin": {
        "moduleDir": "plugin-module"
    },
    "mobile": {
        "moduleDir": "plugin-module"
    },
    "desktop": {
        "moduleDir": "plugin-module"
    }
}
```

Check that your plugin now looks like this:

```
peek-plugin-tutorial
├── peek_plugin_tutorial
│   ├── __init__.py
│   ├── plugin_package.json
```

(continues on next page)

(continued from previous page)

```
│   └── _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 server and the part of the plugin running on the client 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"
```

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 mobile, admin and desktop services.

Developers can use some `index.ts` magic to abstract the layout of their directories. An exmaple 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, oberservable 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 trival a this point, but it becomes more usefull 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 want's 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.

8.9.3 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.

Document Generator

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

The following sections go on to guide the reader to setup Sphinx Document Generator.

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

Documentation Configuration

The build configuration file has already been setup by Synerty.

Create Directories `docs`

The `docs` folder will contain all of the files used to build the documentation. Make sure you add everything in this directory to git.

The `dist` folder will contain all of the generated documentation. These files should not be in git as they are reproducible, see *Build Documentation*

Create directories `docs` and `dist`, run the following command:

```
mkdir -p docs
```

Download file `docs/conf.py`

The `conf.py` file contains the configuration required to build the documentation.

Synerty has created a version of this file that automatically generates the api doc RST files.

Download `conf.py` from synerty-peek/docs/conf.py to `docs/conf.py`.

Modify these values:

```
__project__ = 'Synerty Peek'
__copyright__ = '2016, Synerty'
__author__ = 'Synerty'
__version__ = '#.#.#'
```

At the very end of `conf.py` file, you will see imports and calls to `createApiDocs(peek_plugin_xxx.__file__)` method.

If your plugin will have a python API, then update these two lines to import your plugin, and generate API documentation for it.

From:

```
import peek_plugin_base
createApiDocs(peek_plugin_base.__file__)
```

Example To:

```
try:
    import peek_plugin_tutorial

except ImportError:
    # Otherwise, add the plugin root dir to the import path, for read the docs.
    sys.path.append(os.path.dirname(os.path.dirname(__file__)))
    import peek_plugin_tutorial

createApiDocs(peek_plugin_tutorial.__file__)
```

Otherwise, comment it out.

Required Files

Note: All instructions in this document are relative to the plugin root directory (the one with hyphens), not the plugin python package (the one with underscores).

Add Directory `_static`

The `_static` is required for the doc build.

Create the directory with this command:

```
mkdir docs/_static
```

Add File `.gitkeep`

The `docs/_static/.gitkeep` ensures that the _static directory will exist in git.

Create file `docs/_static/.gitkeep` with no contents.

Create it with this command:

```
touch docs/_static/.gitkeep
```

Add Directory `overview`

The `overview` will contain the the `overview.rst` file and all images that it uses. For now, there are none.

Create the directory with this command:

```
mkdir docs/overview
```

Add File `overview.rst`

The `docs/overview/overview.rst` Should contain a basic overview of the plugin.

Create file `docs/overview/overview.rst` and populate it with the following contents:

```
========
Overview
========

Plugin Objective
----------------
```

(continues on next page)

(continued from previous page)

```
The goal of this plugin is to ...


Plugin Uses
-----------

Possible uses for this plugin are ...


How It Works
------------

This plugin achieves it's functionality by ...
```

Add Directory `api`

The `api` will contain the the `index_api.rst` file and all images that it uses. For now, there are none.

Create the directory with this command:

```
mkdir docs/api
```

Add File `index_api.rst`

The `index_api.rst` contains links to any information useful to other develeopers wanting to leverage this plugin

---

Create file `docs/api/index_api.rst` and populate it with the following contents:

```
.. _api_reference:

=============
API Reference
=============

.. toctree::
    :maxdepth: 2
    :caption: Contents:

    ../api_autodoc/peek_plugin_tutorial/peek_plugin_tutorial
```

Add file `index.rst`

The `index.rst` file will add relations between the single files that the documentation is made of, as well as tables of contents. See *TOC tree*

Note: Add more files to plugin table of contents by addding them after `overview/overview`

Create `index.rst`, and populate it with the following:

```
==================================
{insert plugin name} Documentation
==================================

.. toctree::
    :maxdepth: 3
    :caption: Contents:

    overview/overview
    api/index_api

Indices and tables
==================

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

Add file `rtfd_requirements.txt`

The `rtfd_requirements.txt` is required for building docs on readthedocs.org

Create `docs/rtfd_requirements.txt`, and populate it with the following:

```
pytmpdir
peek_plugin_base
```

Build or Debug

You have created all the configuration files for the documentation generator, let's make a first build of the docs.

You can either *Build Documentation* or *Debug Documentation*

Build Documentation

This section will build the documentation locally as HTML files. From there the developer can copy it somewhere else, etc.

Note: If this is **NOT** the first build of the documentation or you have previously run the *Debug Documentation*, you will need to cleanup the old `dist` files. Run the command `rm -rf dist/*`

Sphinx-build

A build is started with the sphinx-build program, called like this:

```
[ -d dist ] && rm -rf dist
mkdir -p dist/docs
sphinx-build -b html docs/ dist/docs/
```

Note: The -b option selects a builder; in this example Sphinx will build HTML files.

A successful build should look like this:

```
peek@DESKTOP-U08T8NG MINGW64 ~/peek-plugin-tutorial (master)
$ sphinx-build -b html docs/ dist/docs/
Running Sphinx v1.5.3
making output directory...
loading pickled environment... not yet created
building [mo]: targets for 0 po files that are out of date
building [html]: targets for 3 source files that are out of date
updating environment: 3 added, 0 changed, 0 removed
reading sources... [100%] module
looking for now-outdated files... none found
pickling environment... done
checking consistency... done
preparing documents... done
writing output... [100%] module
generating indices... genindex py-modindex
highlighting module code... [100%] peek_plugin_tutorial
writing additional pages... search
copying static files... done
copying extra files... done
dumping search index in English (code: en) ... done
dumping object inventory... done
build succeeded.
```

Open `dist/docs/index.html`

The generated documentation files are in the `dist/docs` folder.

Open `dist/docs/index.html` in a web browser to view the generated documentation.

Debug Documentation

Synerty has written a shell script that runs a sphinx auto build utility. It builds the documentation when a file is modified.

Note: If this is **NOT** the first debugging of the documentation or you have previously run the *Debug Documentation*, you will need to cleanup the old `dist` files. Run the command `rm -rf dist/*`

Download File `watch-docs.sh`

The `watch-docs.sh` script runs an auto building / auto refreshing web server that is fantastic for quick local documentation development.

Download `watch-docs.sh` from synerty-peek/docs/watch-docs.sh to `docs/watch-docs.sh`

Edit `docs/watch-docs.sh` to update the plugin package name.

Change the line:

```
ARGS="$ARGS --watch `modPath 'peek_plugin_base'`"
```

to:

```
ARGS="$ARGS --watch `modPath 'peek_plugin_tutorial'`"
```

Run `watch-docs.sh`

```
cd docs/
./watch-docs.sh
```

In a web browser, go to the following url:

```
http://localhost:8020
```

The `watch-docs.sh` shell script will rebuild the documentation when it see a change in the `docs` folder.

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`.

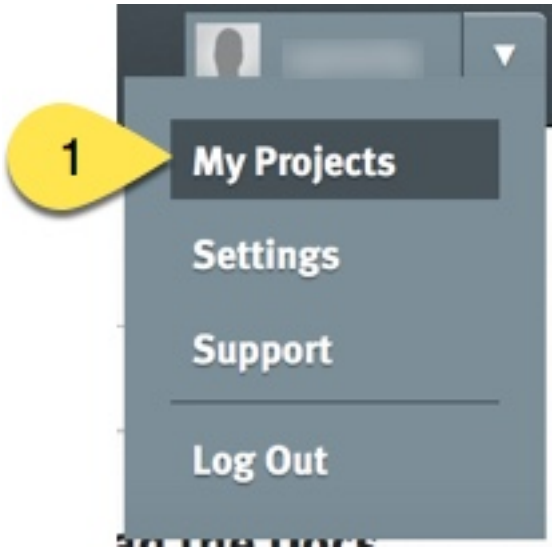
Publish Documentation on readthedocs

Read the Docs hosts documentation, making your documentation fully searchable and easy to find. Your documentation can be imported from versioning control such as Git. Read the Docs support webhooks to build the documentation after the latest code commit.

Create an account on Read the Docs

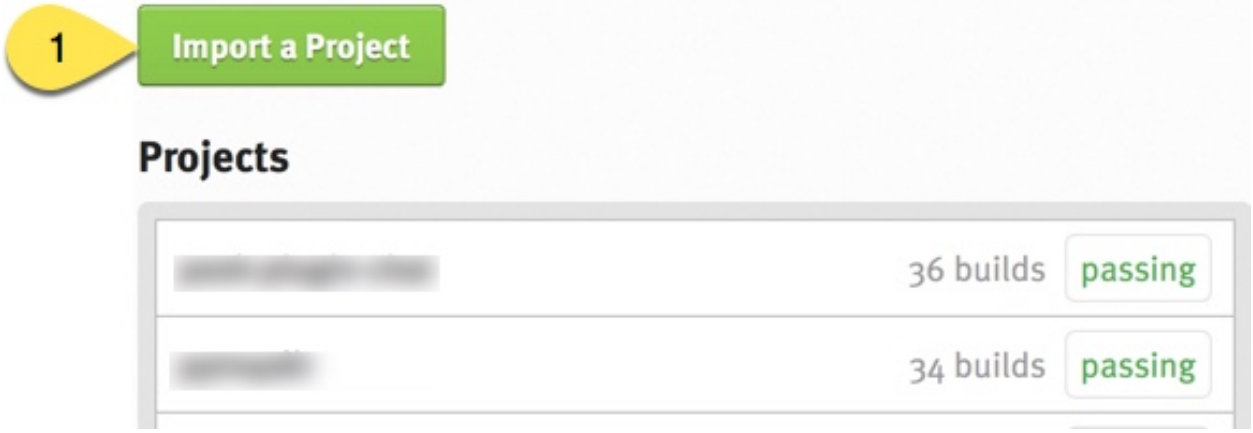
Add a Project to Read the Docs

View "My Projects"



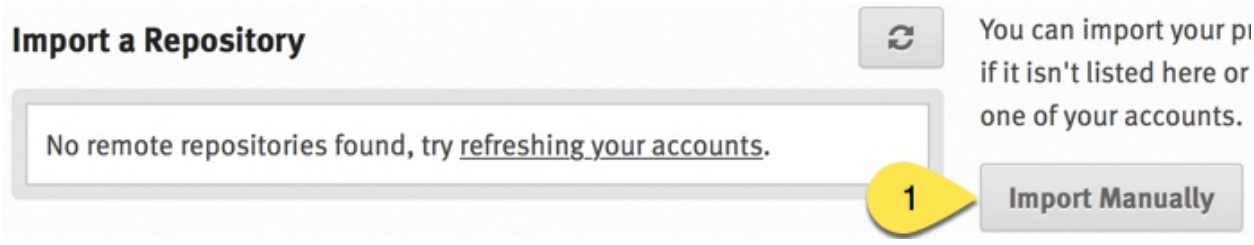
1. Go to the users drop down at the top right and select “My Projects”

Import Projects



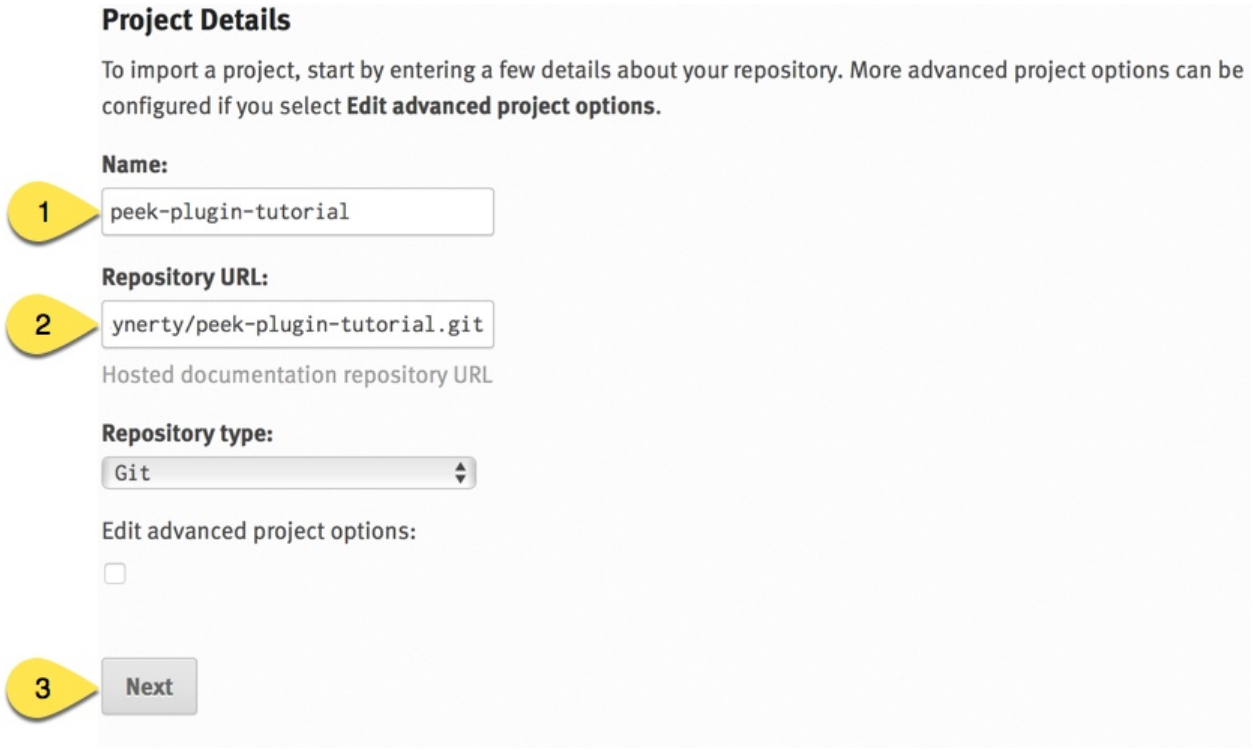
1. Select “Import a Project”

Import Manually



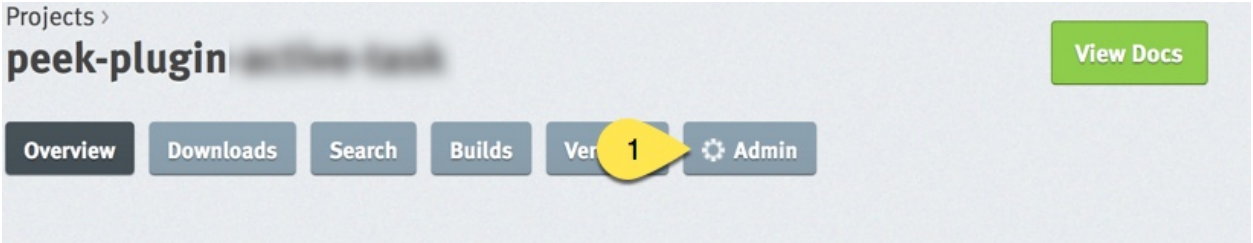
1. Select “Import Manually”

Project Details



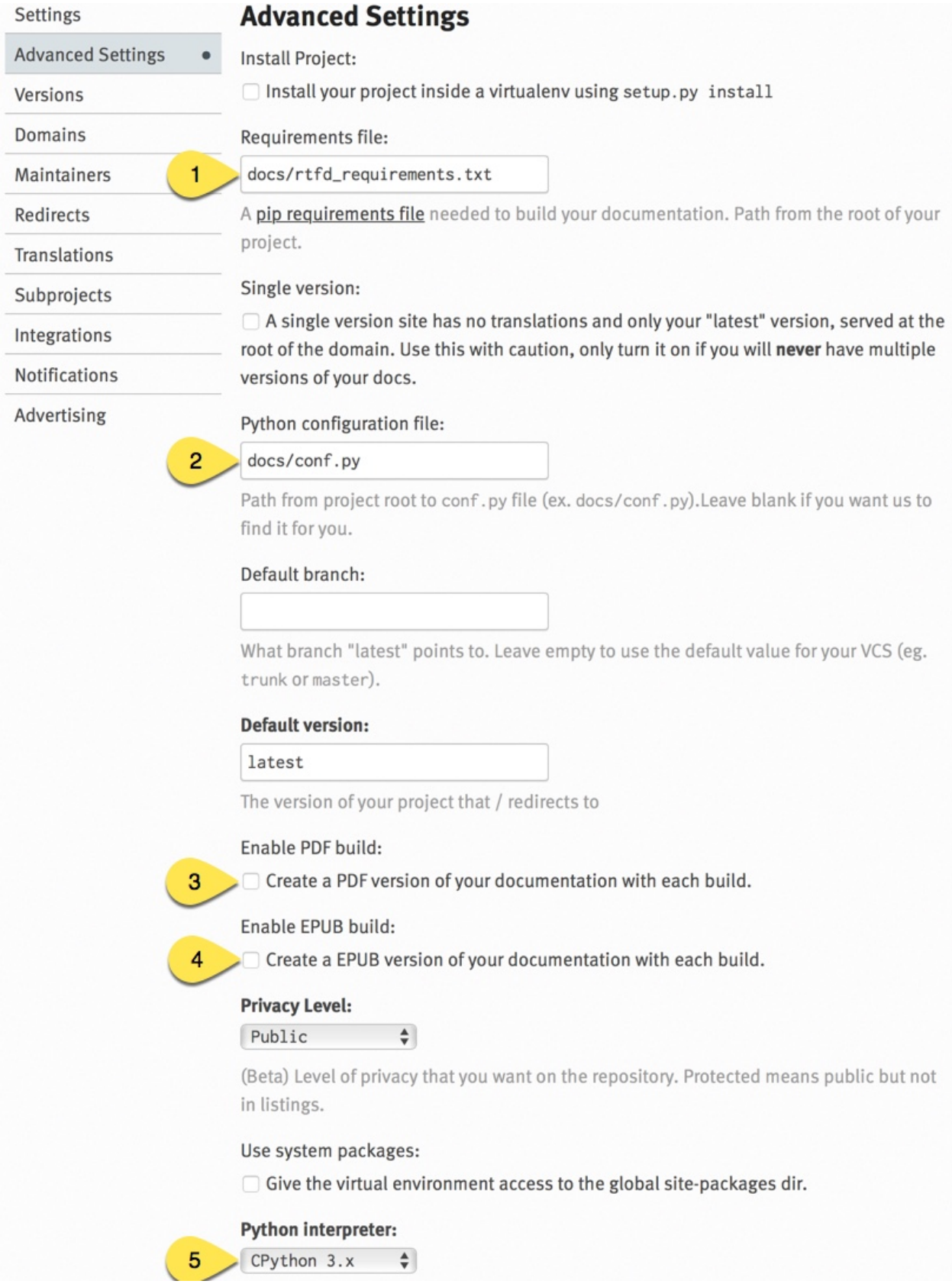
1. Enter the name of the project
2. Enter your git repository location
3. Select “Next”

Project



1. Go to the “Admin” page

Advanced Settings

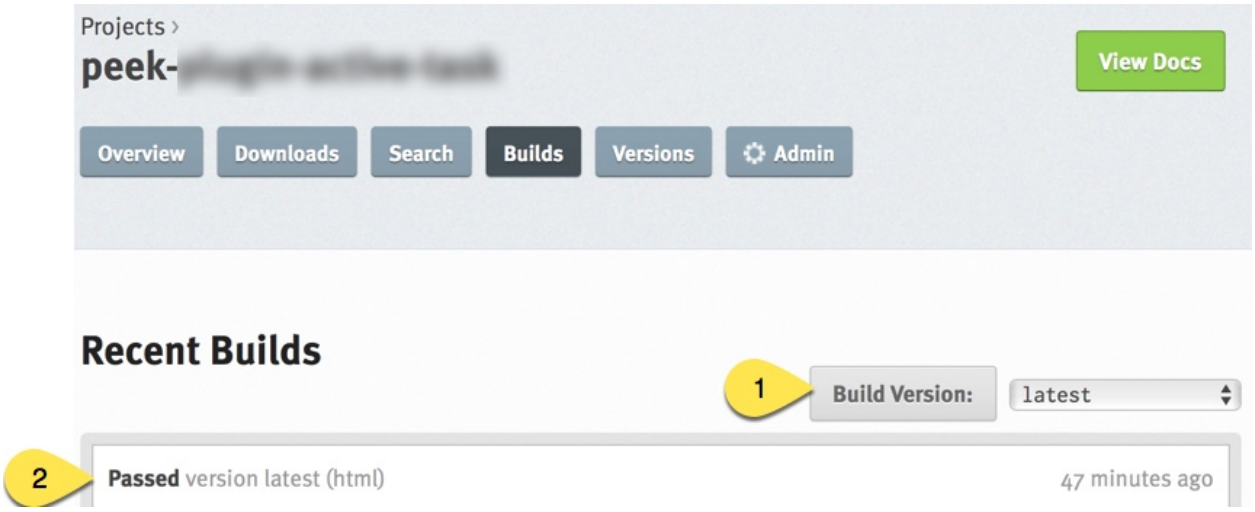


1. Enter the location of the requirements files

2. Enter the location of the sphinx configuration file
3. Uncheck the "Enable PDF Build"
4. Uncheck the "Enable EPUB Build"
5. Select the Python interpreter 3.+

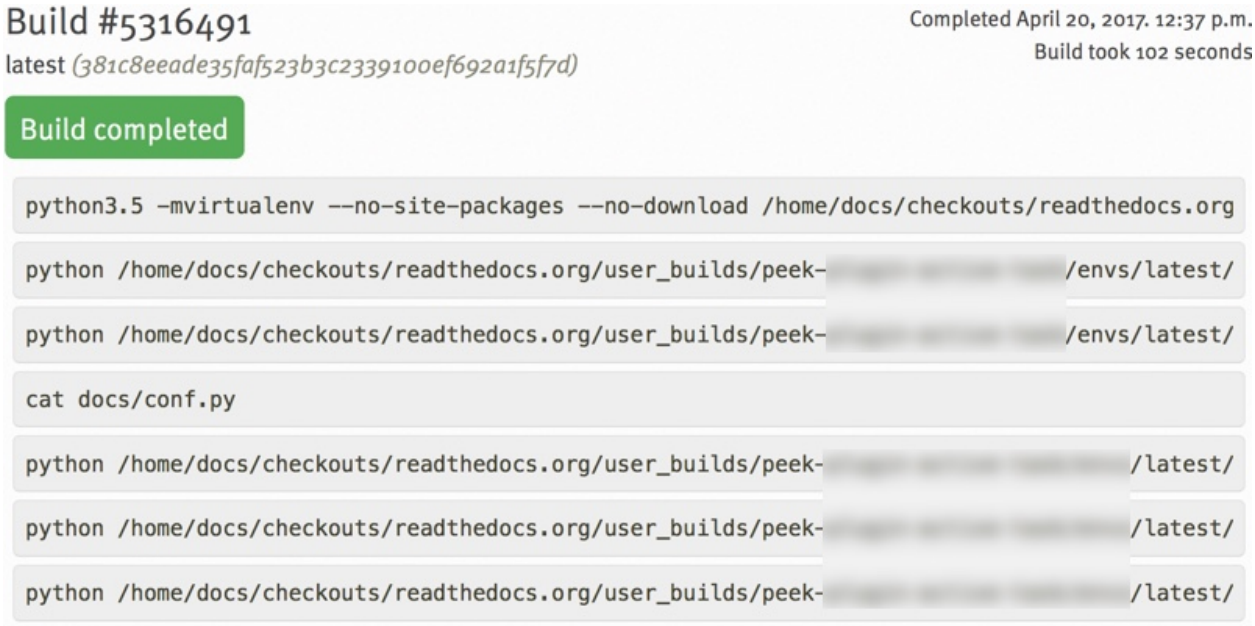
Troubleshooting readthedocs

Build the Documentation



1. Force a Documentation build by selecting "Build Version"
2. View the recent build log

Investigate the build log:



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*.

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.
~~~~~~~~

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.

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`
```

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`
```

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

```
`Synerty <http://www.synerty.com/>`_
```

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
```

Bullets

- First point
- Second point

```
- First point

- Second point
```

Numbered Lists

1. First point
2. Second point

```
#.  First point

#.  Second point
```

Directives

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.

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:: synerty_logo_400x800.png
```

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`
```

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)
```

Docstring Format

This extension `sphinx.ext.atuodoc`, 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/ServerEntryHook.py`, create in the step *Add File ServerEntryHook.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.

    """
    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 it's 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.

        """
```

What Next?

Start developing your own plugins.

8.9.4 Add Server Service

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

Note: Setting up skeleton files for the client, worker and agent services, is identical to the server, generally replace "Server" with the appropriate service name.

The platform loads the plugins python package, and then calls the appropriate **peek{Server}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.

Server File Structure

Add Package `_private/server`

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

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

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

Create directory `peek_plugin_tutorial/_private/server`

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

Commands:

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

Add File `ServerEntryHook.py`

This file/class is the entry point for the plugin on the Server service. When the server 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/server/ServerEntryHook.py` and populate it with the following contents.

```
import logging

from peek_plugin_base.server.PluginServerEntryHookABC import PluginServerEntryHookABC

logger = logging.getLogger(__name__)


class ServerEntryHook(PluginServerEntryHookABC):
    def __init__(self, *args, **kwargs):
        """ Constructor """
        # Call the base classes constructor
        PluginServerEntryHookABC.__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):
        """ 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
```

(continues on next page)

(continued from previous page)

```
        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 Server service loads the plugin, it first calls the **`peekServerEntryHook()`** method from the **`peek_plugin_tutorial`** package.

The **`peekServerEntryHook()`** method returns the Class that the server 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 it's root python package.

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

```
from peek_plugin_base.server.PluginServerEntryHookABC import PluginServerEntryHookABC
from typing import Type


def peekServerEntryHook() -> Type[PluginServerEntryHookABC]:
    from ._private.server.ServerEntryHook import ServerEntryHook
    return ServerEntryHook
```

Edit `plugin_package.json`

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

Edit the file `peek_plugin_tutorial/plugin_package.json`:

1. Add **"server"** to the requiresServices section so it looks like

   ```
   "requiresServices": [
       "server"
   ]
   ```

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

   ```
   "server": {
   }
   ```

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

Here is an example

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

}
```

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

Running on the Server Service

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

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

Edit `~/peek-server.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 server, you should see your plugin load.

```
peek@peek:~$ run_peek_server
...
DEBUG peek_plugin_tutorial._private.server.ServerEntryHook:Loaded
DEBUG peek_plugin_tutorial._private.server.ServerEntryHook:Started
...
```

8.9.5 Add Client Service

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

Client File Structure

Add Package `_private/client`

Create directory `peek_plugin_tutorial/_private/client`

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

Commands:

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

Add File `ClientEntryHook.py`

Create the file `peek_plugin_tutorial/_private/client/ClientEntryHook.py` and populate it with the following contents.

```
import logging

from peek_plugin_base.client.PluginClientEntryHookABC import PluginClientEntryHookABC

logger = logging.getLogger(__name__)


class ClientEntryHook(PluginClientEntryHookABC):
    def __init__(self, *args, **kwargs):
        """ Constructor """
        # Call the base classes constructor
        PluginClientEntryHookABC.__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")
```

(continues on next page)

(continued from previous page)

```
    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.client.PluginClientEntryHookABC import PluginClientEntryHookABC
from typing import Type


def peekClientEntryHook() -> Type[PluginClientEntryHookABC]:
    from ._private.client.ClientEntryHook import ClientEntryHook
    return ClientEntryHook
```

Edit `plugin_package.json`

Edit the file `peek_plugin_tutorial/plugin_package.json`:

1. Add **"client"** to the requiresServices section so it looks like

```
"requiresServices": [
    "client"
]
```

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

```
"client": {
}
```

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

Here is an example

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

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

Running on the Client Service

Edit `~/peek-client.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 Client Service will also create the file.

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

```
peek@peek:~$ run_peek_client
...
DEBUG peek_plugin_tutorial._private.client.ClientEntryHook:Loaded
DEBUG peek_plugin_tutorial._private.client.ClientEntryHook:Started
...
```

8.9.6 Add Agent Service

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

Agent 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 initialiastion steps here.

        """
        logger.debug("Loaded")
```

(continues on next page)

(continued from previous page)

```
    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.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 to load.

Running on the Agent Service

Edit `~/peek-agent.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*. Running the Agent Service will also create the file.

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

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

8.9.7 Add Storage Service

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

Peek Storage connects to a database server, provides each plugin it's 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 Server services.

The Storage schema upgrades are managed by the Server service.

Note: The Server service must be enabled to use the Storage service.

Storage 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
```

(continues on next page)

(continued from previous page)

```
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, server, worker
    and client 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 it's 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"
    ],
    ...
    "storage": {
    }
}
```

Edit File `ServerEntryHook.py`

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

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

Edit the file `peek_plugin_tutorial/_private/server/ServerEntryHook.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.server.PluginServerStorageEntryHookABC import␣
↪PluginServerStorageEntryHookABC
```

2. Add **PluginServerStorageEntryHookABC** to the list of classes **"ServerEntryHook"** inherits

```
class ServerEntryHook(PluginServerEntryHookABC, PluginServerStorageEntryHookABC):
```

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.server.PluginServerStorageEntryHookABC import \
    PluginServerStorageEntryHookABC


# Added inherited class, step2
class ServerEntryHook(PluginServerEntryHookABC, PluginServerStorageEntryHookABC):


    def load(self) -> None:
        # 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 `ClientEntryHook.py`

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

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

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

Edit the file `peek_plugin_tutorial/_private/client/ClientEntryHook.py`

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

```
from peek_core_device._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_core_device._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 ClientEntryHook.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 ClientEntryHook.py*.

Add File `alembic.ini`

The `alembic.ini` file is the first file Alembic laods, it tells Alembic how to connect to the database and where it's "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 `postgresql://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 server, 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(50))
    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/server/ServerEntryHook.py`

Add the following import up the top of the file:

```
from peek_plugin_pof_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()
```

8.9.8 Add Admin Service

The admin service is the admin user interface. This is known as the "frontend" in web terminology. The backend for the Admin service is the Server 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 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'
})
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 service:

1. Add **"admin"** to the requiresServices section so it looks like

```
"requiresServices": [
    "admin"
]
```

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

```
"admin": {
    "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 Service

The Peek Server service provides the web service that serves the admin angular application.

The Peek Server service takes care of combining all the plugin files into the build directories in the peek_admin package. We will need to restart Peek Server for it to include our plugin in the admin UI.

See *Developing With The Frontends* for more details.

Check File `~/peek-server.home/config.json`

Check the `~/peek-server.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
```

(continues on next page)

(continued from previous page)

```
    },
    ...
}
```

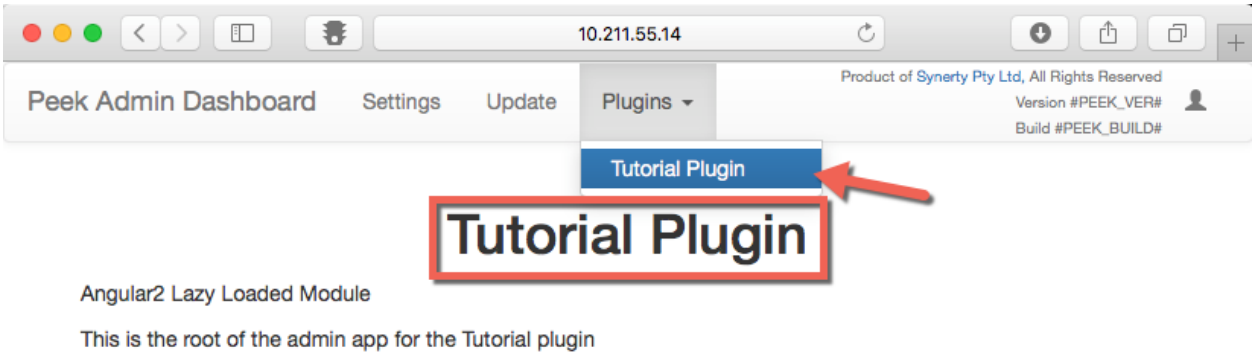
Run `run_peek_server`

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

```
peek@peek:~$ run_peek_server
...
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_server`**.

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



8.9.9 Add Mobile Service

The mobile service is for the users. It's the interface designed for mobile devices.

The Mobile service is known as the "frontend" in web terminology. The backend for the Mobile service is the Client service.

The Peek Mobile Service has two builds, a NativeScript build and a web build.

In this document, we'll add the start of both the mobile and web 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.

Mobile File Structure

Add Directory `mobile-app`

The `mobile-app` directory will contain the plugins the mobile 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/mobile-app`

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/mobile-app/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 mobile app for the Tutorial plugin</p>
</div>
```

Add File `tutorial.component.ns.html`

The `tutorial.component.ns.html` file is the default NativeScript **view** for the Angular component `tutorial.component.ts`.

This is standard NativeScript, it's actually XML, but the IDE provides better assistance and autocomplete for the angular directives if the file ends in .html.

NativeScript doesn't run in a browser. It's a native mobile app running NodeJS, and many special bindings that allow JavaScript/TypeScript to call native Android and iOS libraries.

Using this technique, NativeScript can call native UI libraries, allowing developers to write code with Javascript, yet still have smooth, responsive and offline user interfaces.

See Adding UI elements , for the NativeScript introduction of their views.

Important: NativeScript is nothing like HTML, It's important to understand this. The only common element is that they both have Angular directives.

NativeScript has a completely different layout system, there are no <p> tags, and plain text outside of tags won't just show up in the app.

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

```
<StackLayout class="p-20" >
    <Label text="Tutorial Plugin" class="h1 text-center"></Label>
    <Label text="Angular2 Lazy Loaded Module" class="h3 text-center"></Label>
    <Label text="This is the root of the mobile app for the Tutorial plugin"
        class="h3 text-center"></Label>
</StackLayout>
```

Add File `tutorial.component.ts`

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

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

Create the file `peek_plugin_tutorial/_private/mobile-app/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/mobile-app/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 a small abstraction library to switch between nativescript and web
import {PeekModuleFactory} from "@synerty/peek-mobile-util/index.web";

// 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 mobile interface has a home screen with apps on it, this icon will be the tutorial plugins app icon.

Create directory `peek_plugin_tutorial/_private/mobile-assets`

Download this plugin app icon TutorialExampleIcon.png to `peek_plugin_tutorial/_private/mobile-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 mobile service:

1. Add **"mobile"** to the requiresServices section so it looks like

```
"requiresServices": [
    "mobile"
]
```

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

```
"mobile": {
    "showHomeLink": true,
    "appDir": "_private/mobile-app",
    "appModule": "tutorial.module#TutorialModule",
    "assetDir": "_private/mobile-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": [
        ...
        "mobile"
    ],
    ...
    "mobile": {
        "showHomeLink": true,
        "appDir": "_private/mobile-app",
        "appModule": "tutorial.module#TutorialModule"
    }
}
```

Running the Mobile Web App

The Peek Client service provides the web service that serves the mobile angular web app.

The Peek Client service takes care of combining all the plugin files into the build directories in the peek_mobile package. We will need to restart Peek Client for it to include our plugin in the mobile UI.

See *Developing With The Frontends* for more details.

Check File `~/peek-client.home/config.json`

Check the `~/peek-client.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
    },
    ...
}
```

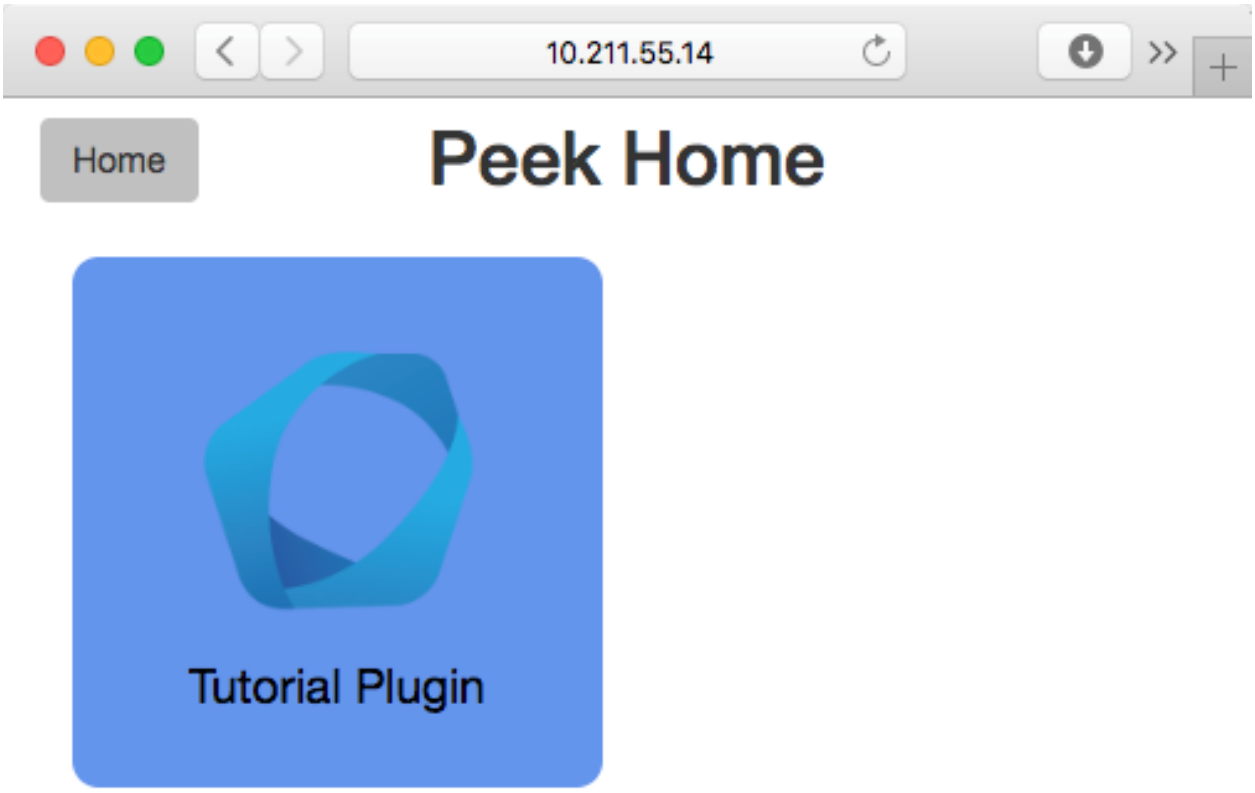
Run `run_peek_client`

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

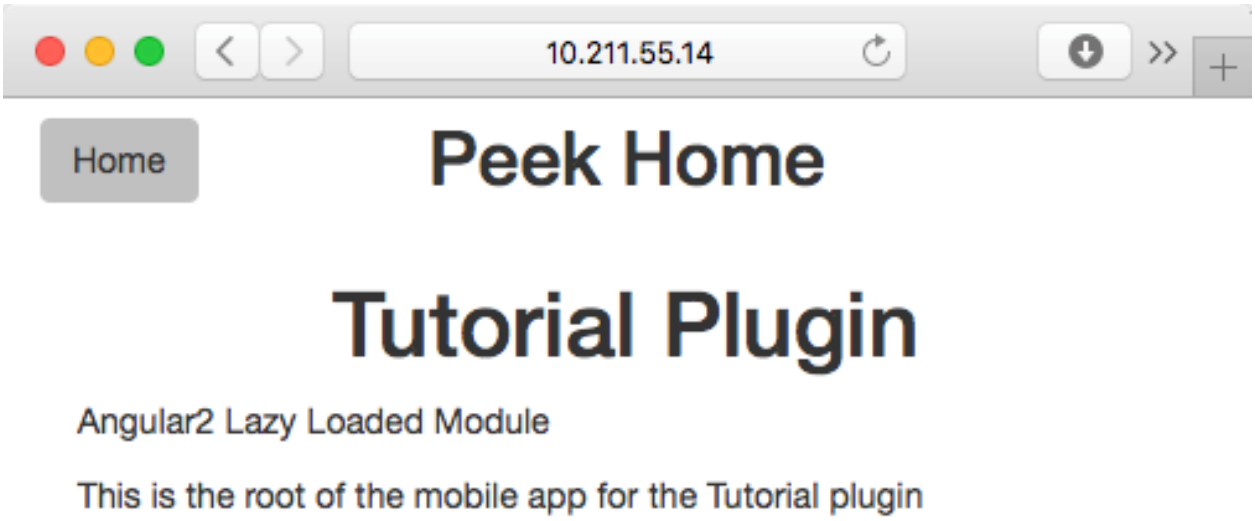
```
peek@peek:~$ run_peek_client
...
INFO peek_platform.frontend.WebBuilder:Rebuilding frontend distribution
...
INFO txhttputil.site.SiteUtil:Peek Client is alive and listening on http://10.211.55.
↪14:8000
...
```

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

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



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



Running the Mobile NativeScript App

The Peek Client service provides the websocket that the NativeScript app uses. The NativeScript application uses all the same code to run as the Web App, The only difference is the view file.

With Peek, you can develop a web app and a native app, with little more effort.

The Peek Client service takes care of combining all the plugin files into the build directories in the peek_mobile package. We will need to restart Peek Client for it to include our plugin in the mobile UI.

See *Developing With The Frontends* for more details.

Check File `~/peek-client.home/config.json`

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

1. Ensure **frontend.nativescriptBuildPrepareEnabled** is set to **true**, with no quotes

---

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

Example:

```
{
    ...
    "frontend": {
        ...
        "nativescriptBuildPrepareEnabled": true,
    },
    ...
}
```

Run `run_peek_client`

Run the peek client, The NativeScript will be offline with out it.

```
peek@peek:~$ run_peek_client
...
INFO txhttputil.site.SiteUtil:Peek Client is alive and listening on http://10.211.55.
↪14:8010
...
```

tns run android

This section runs the NativeScript app on an Emulator, or a real Device. NativeScript must be installed before proceeding.

- *Setup Nativescript Windows*
- *Setup Nativescript Debian*

See Running NativeScript Apps for some details on **`tns run`**.

We use the Android platform to test the apps as it runs on Windows, Mac and Linux.

In this example, NativeScript will run in all connected devices and emulators, or it will start an emulator.

Change directory to the build-ns directory under the peek_mobile python package. Run the following in bash to get the path of the build-ns directory:

```
python << EOPY
import os.path as p
import peek_mobile
print("Using peek_mobile version %s, located at:" % peek_mobile.__version__)
print("    " + p.join(p.dirname(peek_mobile.__file__), 'build-ns'))
EOPY
```

Now CD to that directory, Example:

```
cd /home/peek/project/peek-mobile/peek_mobile/build-ns
```

Check the devices that are connected, if one isn't connected, NativeScript will try to start the standard android emulator.

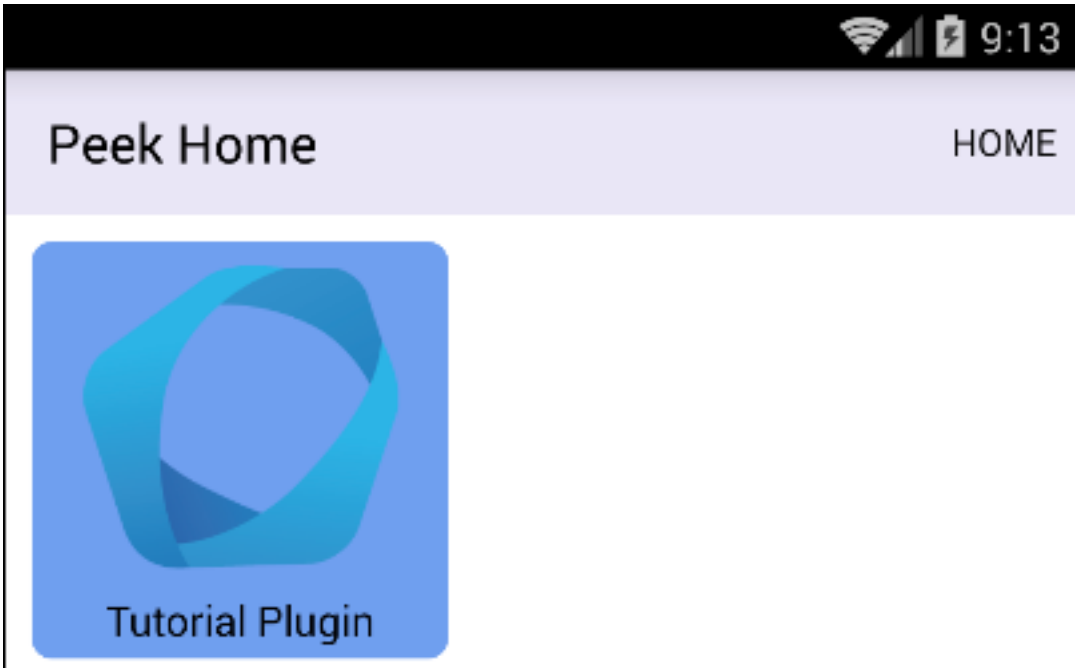
```
peek@peek:~/project/peek-mobile/peek_mobile/build-ns$ tns device list
iTunes is not available for this operating system. You will not be able to work with␣
↪connected iOS devices.
─────────────────────────────────────────────────────────────────────────────
│ # │ Device Name │ Platform │ Device Identifier │ Type     │ Status    │
│ 1 │ vbox86p     │ Android  │ emulator-5554     │ Emulator │ Connected │
└───────────────────────────────────────────────────────────────────────────
```

Run **`tns run android`**

```
tns run android
```

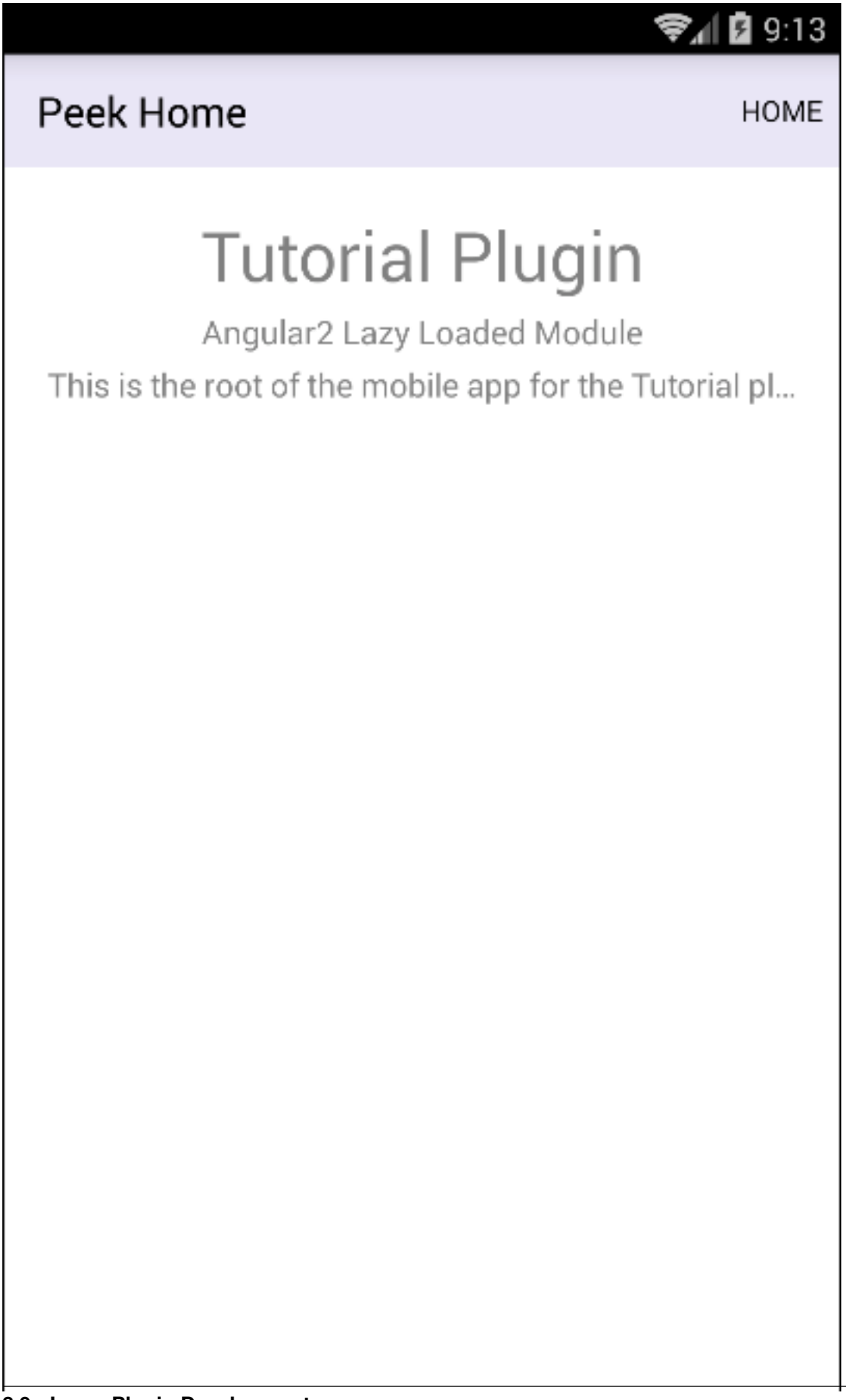
It will take up to two minutes to build, install and run.

You should see the app start, with a splash screen. Then you will see your plugin on the home screen. Touch the App/Plugin icon.


9:13
Peek Home
HOME
Tutorial Plugin

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

And if this is your first Native mobile app, Congratulations, the sky is your limit.


9:13
Peek Home
HOME
Tutorial Plugin
Angular2 Lazy Loaded Module
This is the root of the mobile app for the Tutorial pl...

8.9.10 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 services 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, client, server 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 date1
    dict1 = TupleField(defaultValue=dict)

    #:  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 `ServerEntryHook.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/server/ServerEntryHook.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 `ClientEntryHook.py`

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

Note: This service was add earlier in this tutorial, see *Add Client Service*

Edit file `peek_plugin_tutorial/_private/client/ClientEntryHook.py` file, apply the same edits from step *Edit File ServerEntryHook.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 ServerEntryHook.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 ServerEntryHook.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
```

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
    dict1 : {};

    //  Description of array1
    array1 : any[];

    //  Description of date1
    date1 : Date;

    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 date1
    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";


@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() {
```

(continues on next page)

(continued from previous page)

```
        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.

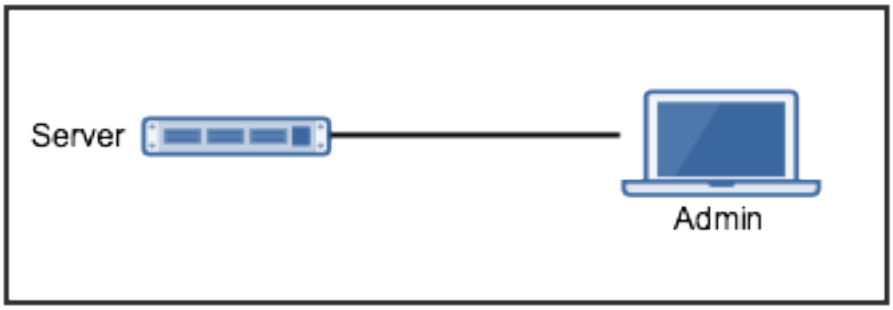
8.9.11 Add Tuple Loader

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 Server 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 server and admin parts of the plugin.

Advantages

1. Easily edit table data.

Disadvantages

1. Not suitable for multiple users.

Server 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/server/admin_backend` package, with the commands

```
mkdir peek_plugin_tutorial/_private/server/admin_backend
touch peek_plugin_tutorial/_private/server/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/server/admin_backend/__init__.py` Add the following:

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

Edit File `ServerEntryHook.py`

Now, we need to create and destroy our `admin_backend` handlers when the Server 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/server/ServerEntryHook.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 initialiastion steps here.

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

Test Python Services

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

StringInt Server 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 Server 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 hander
```

(continues on next page)

(continued from previous page)

```
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, retreiveAll=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/server/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>
            </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>
```

(continues on next page)

(continued from previous page)

```
            </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 ComponentLifecycleEventEmitter, 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 {Ng2BalloonMsgService} from "@synerty/ng2-balloon-msg";
import {
    extend,
    VortexService,
    ComponentLifecycleEventEmitter,
    TupleLoader
} from "@synerty/vortexjs";
import {StringIntTuple,
    tutorialFilt
} from "@peek/peek_plugin_tutorial/_private";


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

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

(continues on next page)

(continued from previous page)

```
    loader: TupleLoader;

    constructor(private balloonMsg: Ng2BalloonMsgService,
                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;

        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));
    }
```

(continues on next page)

(continued from previous page)

```
}
```

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 `</ul>` 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 Server 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 Server 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, retreiveAll=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/server/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>
                </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"
```

(continues on next page)

(continued from previous page)

```
                                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 {Ng2BalloonMsgService} from "@synerty/ng2-balloon-msg";
import {
    ComponentLifecycleEventEmitter,
    extend,
    TupleLoader,
    VortexService
} from "@synerty/vortexjs";
import {SettingPropertyTuple, tutorialFilt} from "@peek/peek_plugin_tutorial/_private
↪";


@Component({
```

(continues on next page)

(continued from previous page)

```
    selector: 'pl-tutorial-edit-setting',
    templateUrl: './edit.component.html'
})
export class EditSettingComponent extends ComponentLifecycleEventEmitter {
    // 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: Ng2BalloonMsgService,
                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 Server 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.

8.9.12 Add Offline Storage

Outline

The Offline Storage is used by the Mobile and Desktop services. It provides an easy way to save and load tuples in the devices

This data can be accessed offline, or loaded before the Client service has responded to a request for data.

In this document, we setup a provider for the Angular Service.

Mobile 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/mobile-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
        },
        ...
    ]
})
export class TutorialModule {

}
```

Complete.

The tutorial plugin is now setup to use the TupleOffline serivce. This service is used by `TupleActionPushOfflineService` and `TupleDataOfflineObserverService` services.

A developer can use the `TupleOfflineStorageService` service if they wish but thats out side the scope of this tutorial.

8.9.13 Add Observables

Outline

In this document, we setup the Tuple Observable from VortexJS. The Mobile and Desktop services use this to request and receive data updates from the Service service.

We'll use the term "devices" interchangeably with Mobile/Desktop.

This is a one directional data flow once the initial request has been made, the Server will send updates to the Mobile/Desktop with out the Mobile/Desktop services polling for it.

In the example setup, the Client proxies Observable requests/responses between the Server and Mobile/Desktop devices. The Proxy on the Client is aware of all the Mobile/Desktop devices that want to observe the data, the Server only knows that the Client is observing the data.

Note: The Mobile/Desktop devices don't and can't talk directly to the Server service.

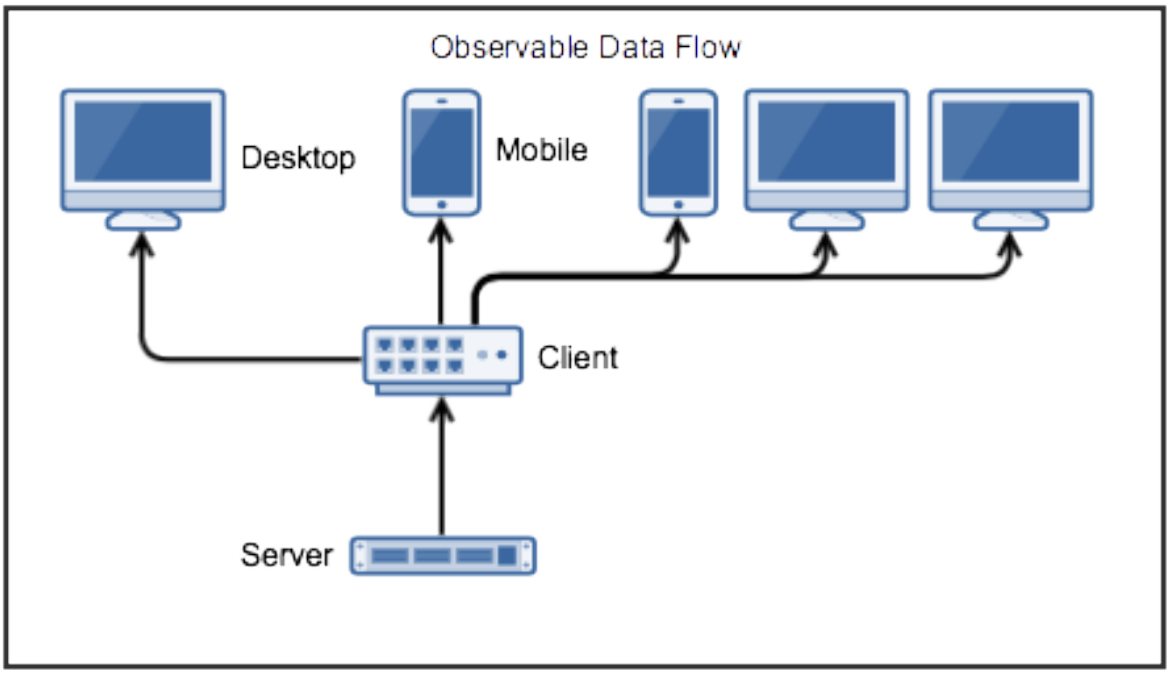
The `TupleDataObservableHandler` class provides the "observable" functionality, It receives request for data by being sent a `TupleSelector`. The TupleSelector describes the Tuple type and some conditions of the data the observer wants.

Advantages

1. Instant and efficient data updates, data immediately sent to the devices with out 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 Server side Tuple Provider
3. Tell the Admin TupleLoader to notifyDeviceInfo the Observable when it makes updates.
4. Add a new Mobile 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 `ServerEntryHook.py`

We need to update `ServerEntryHook.py`, it will initialise the observable object when the Plugin is started.

Edit the file `peek_plugin_tutorial/_private/server/ServerEntryHook.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 Server service is setup now. We'll add a TupleProvider later.

Client Service Setup

Add File `DeviceTupleDataObservableProxy.py`

The `DeviceTupleDataObservableProxy.py` creates the Observable Proxy. This class is responsible for proxying obserable data between the devices and the Server.

It reduces the load on the server, providing the ability to create more Client 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/client/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 `ClientEntryHook.py`

We need to update `ClientEntryHook.py`, it will initialise the observable proxy object when the Plugin is started.

Edit the file `peek_plugin_tutorial/_private/client/ClientEntryHook.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())
   ```

Mobile Service Setup

Now we need to edit the Angular module in the mobile-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/mobile-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.val1 == val1)
                .all()
            )

            # Create the vortex message
            return Payload(filt, tuples=tasks).toVortexMsg()

        finally:
```

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 Mobile/Desktop 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.OrmCrudHandler import OrmCrudHandlerExtension
```

Insert the following class, after the class definition of `class __CrudHandeler`

```
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 :code:‘‘ method, insert this line just before the return `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 notifyDeviceInfo it's observers when an admin updates the StringInt data.

Add Mobile View

Finally, lets add a new component to the mobile screen.

Add Directory `string-int`

The `string-int` directory will contain the Angular component and views for our stringInt page.

Create the diretory `peek_plugin_tutorial/_private/mobile-app/string-int` with the command:

```
mkdir peek_plugin_tutorial/_private/mobile-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/mobile-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.ns.html`

The `string-int.component.ns.html` file is the NativeScript **view** for the Angular component `string-int.component.ts`.

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

```
<StackLayout class="p-20" >
    <Button text="Back to Main" (tap)="mainClicked()"></Button>

    <GridLayout columns="4*, 1*" rows="auto" width="*">
        <Label class="h3" col="0" text="String"></Label>
        <Label class="h3" col="1" text="Int"></Label>
    </GridLayout>

    <ListView [items]="stringInts">
        <template let-item="item" let-i="index" let-odd="odd" let-even="even">
            <StackLayout [class.odd]="odd" [class.even]="even" >
                <GridLayout columns="4*, 1*" rows="auto" width="*">
                    <!-- String -->
                    <Label class="h3 peek-field-data-text" row="0" col="0"
                           textWrap="true"
                           [text]="item.string1"></Label>
```

(continues on next page)

(continued from previous page)

```
                    <!-- Int -->
                    <Label class="h3 peek-field-data-text" row="0" col="1"
                           [text]="item.int1"></Label>

                </GridLayout>
            </StackLayout>
        </template>
    </ListView>
</StackLayout>
```

Add File `string-int.component.ts`

The `string-int.component.ts` is the Angular Component that drives both Web and NativeScript views

This will be another route within the Tutorial plugin.

Create the file `peek_plugin_tutorial/_private/mobile-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 {
    ComponentLifecycleEventEmitter,
    TupleDataObserverService,
    TupleSelector
} from "@synerty/vortexjs";

@Component({
    selector: 'plugin-tutorial-string-int',
    templateUrl: 'string-int.component.mweb.html',
    moduleId: module.id
})
export class StringIntComponent extends ComponentLifecycleEventEmitter {

    stringInts: Array<StringIntTuple> = [];

    constructor(private tupleDataObserver: TupleDataObserverService,
                private router: Router) {
        super();

        // Create the TupleSelector to tell the obserbable 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;
```

(continues on next page)

(continued from previous page)

```
            });

        // unsubscribe when this component is destroyed
        // This is a feature of ComponentLifecycleEventEmitter
        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/mobile-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
       ], ...
   ```

At this point Mobile 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/mobile-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>
```

Edit File `tutorial.component.ns.html`

Edit the NativeScript XML view file, `tutorial.component.ns.html` and insert a button that will change Angular Routes to our new component.

Edit file `peek_plugin_tutorial/_private/mobile-app/tutorial.component.ns.html`, Insert the following just before the closing `</StackLayout>` tag:

```
<Button text="String Ints"
        [nsRouterLink]="['/peek_plugin_tutorial/stringint']"></Button>
```

Testing

1. Open mobile Peek 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 mobile all 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 client
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 (Mobile/Desktop service) from the observable (Client 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/mobile-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 Client 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, *Add File TutorialTuple.ts*.

Add a Server service tuple provider, *Add Tuple Provider*

Then, add the Mobile, Desktop or Admin side, add the views and Angular component, *Add Mobile View*.

8.9.14 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.

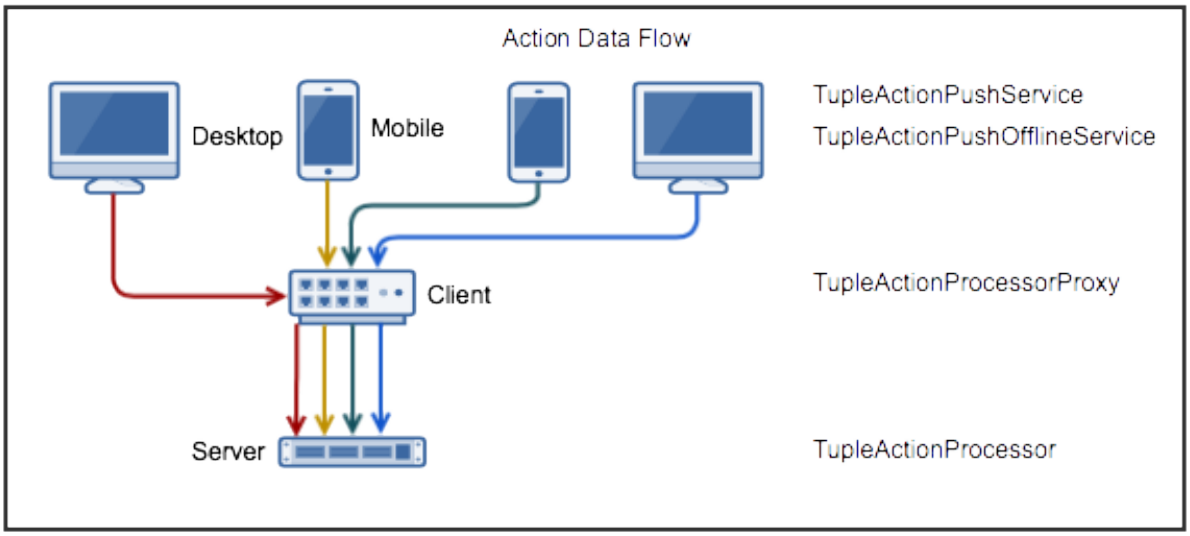
The Observable then ensures that all users watching the data are updated immediately, Keeping all users working with the latest data as TupleActions processed.

This helps avoid issues, such as one users update overwriting another users 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 Client 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 Server 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 Client service that passes actions onto the Server service for processing.

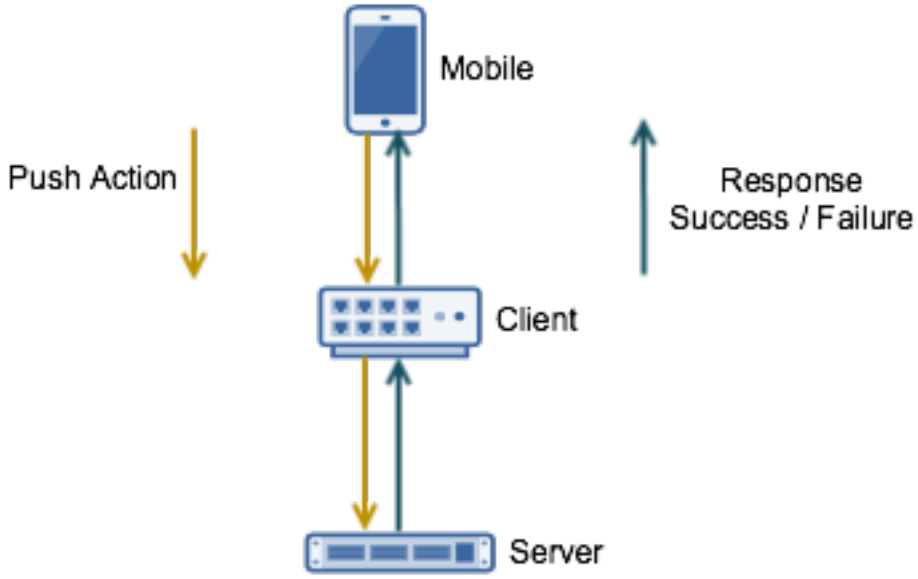
Unlike the Observable, the TupleAction Client proxy passes every action onto the Server service, waits for a response from the Server service then sends that back to the Mobile or Desktop device.

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 Desktop/Mobile device.

The `TupleActionPushOfflineService` will continually retry until it receives either a success or failure response from the Client service.

Note: The Mobile/Desktop devices don't and can't talk directly to the Server 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 Server 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 Server 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):
    __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 {
    public 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";
```

Server 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/server/controller` package, with the commands

```
mkdir peek_plugin_tutorial/_private/server/controller
touch peek_plugin_tutorial/_private/server/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/server/controller/MainController.py` and populate it with the following contents.

```
import logging

from twisted.internet.defer import Deferred
from txhttputil.util.DeferUtil import deferToThreadWrap

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

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:
```

(continues on next page)

(continued from previous page)

```
        if isinstance(tupleAction, AddIntValueActionTuple):
            return self._processAddIntValue(tupleAction)

        if isinstance(tupleAction, StringCapToggleActionTuple):
            return self._processCapToggleString(tupleAction)

        raise NotImplementedError(tupleAction.tupleName())

    @deferToThreadWrap
    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()

    @deferToThreadWrap
    def _processAddIntValue(self, action: AddIntValueActionTuple):
        try:
            # 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(), {})
```

(continues on next page)

(continued from previous page)

```
            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/server/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 `ServerEntryHook.py`

We need to update `ServerEntryHook.py`, it will initialise the `MainController` and `TupleActionProcessor` objects.

Edit the file `peek_plugin_tutorial/_private/server/ServerEntryHook.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 Server service is setup now.

Client Service Setup

Add File `DeviceTupleProcessorActionProxy.py`

The `DeviceTupleProcessorActionProxy.py` creates the Tuple Action Processoe Proxy. This class is responsible for proxying action tuple data between the devices and the Server.

Create the file `peek_plugin_tutorial/_private/client/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 `ClientEntryHook.py`

We need to update `ClientEntryHook.py`, it will initialise the tuple action proxy object when the Plugin is started.

Edit the file `peek_plugin_tutorial/_private/client/ClientEntryHook.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())
   ```

Mobile Service Setup

Now we need to edit the Angular module in the mobile-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/mobile-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/mobile-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/mobile-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>
```

(continues on next page)

(continued from previous page)

```
        </tr>
        </tbody>
    </table>
</div>
```

Edit File `string-int.component.ns.html`

Edit the NativeScript XML view file, `string-int.component.ns.html` and insert buttons that will change initiate the created tuple actions.

Edit the file `peek_plugin_tutorial/_private/mobile-app/string-int/string-int.component.ns.html` and populate it with the following contents.

```
<StackLayout class="p-20">
    <Button text="Back to Main" (tap)="mainClicked()"></Button>

    <GridLayout columns="4*, 1*" rows="auto" width="*">
        <Label class="h3" col="0" text="String"></Label>
        <Label class="h3" col="1" text="Int"></Label>
    </GridLayout>

    <ListView [items]="stringInts">
        <template let-item="item" let-i="index" let-odd="odd" let-even="even">
            <StackLayout [class.odd]="odd" [class.even]="even">
                <GridLayout columns="4*, 1*" rows="auto" width="*">
                    <!-- String -->
                    <Label class="h3 peek-field-data-text" row="0" col="0"
                           textWrap="true"
                           [text]="item.string1"></Label>

                    <!-- Int -->
                    <Label class="h3 peek-field-data-text" row="0" col="1"
                           [text]="item.int1"></Label>

                </GridLayout>
                <Button text="Toggle Caps" (tap)="toggleUpperClicked(item)"></Button>
                <Button text="Increment Int" (tap)="incrementClicked(item)"></Button>
                <Button text="Decrement Int" (tap)="decrementClicked(item)"></Button>
            </StackLayout>
        </template>
    </ListView>
</StackLayout>
```

Testing

1. Open mobile Peek 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 server 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 client
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 (Mobile/Desktop service) from the observable (Client 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/mobile-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 Client service is offline.

8.9.15 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 server service for example.

In this document, we go through using the Vortex RPC to simplify communications between the server and agent service.

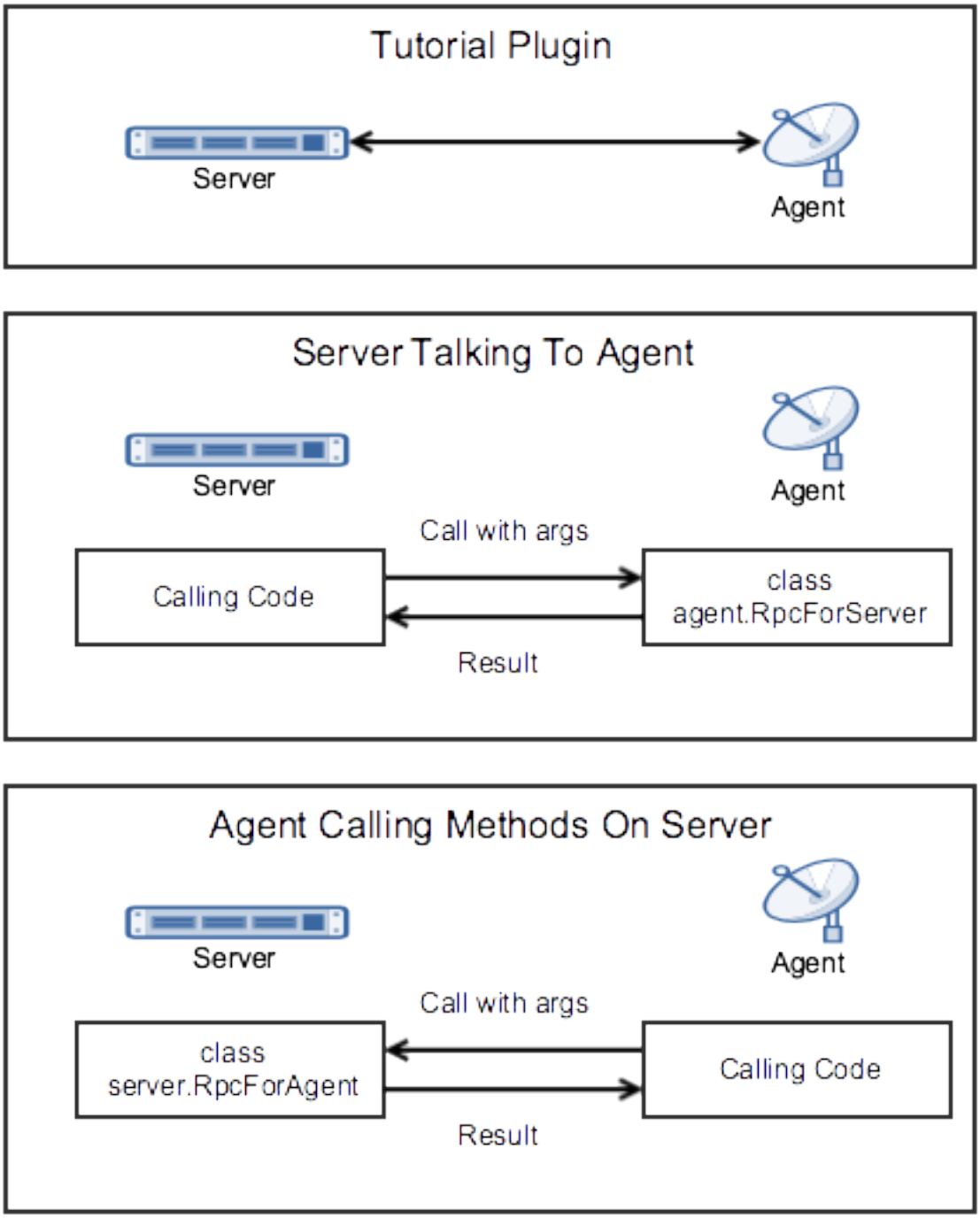
What is RPC

RPC stands for Remote Procedure Call, essentially is 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 Server service. These are completely separate processes, so you can't just call a method defined in the server service from the agent service.

Vortex RPC provides wrappers that make it easy to define procedures in one service, and call them from another.

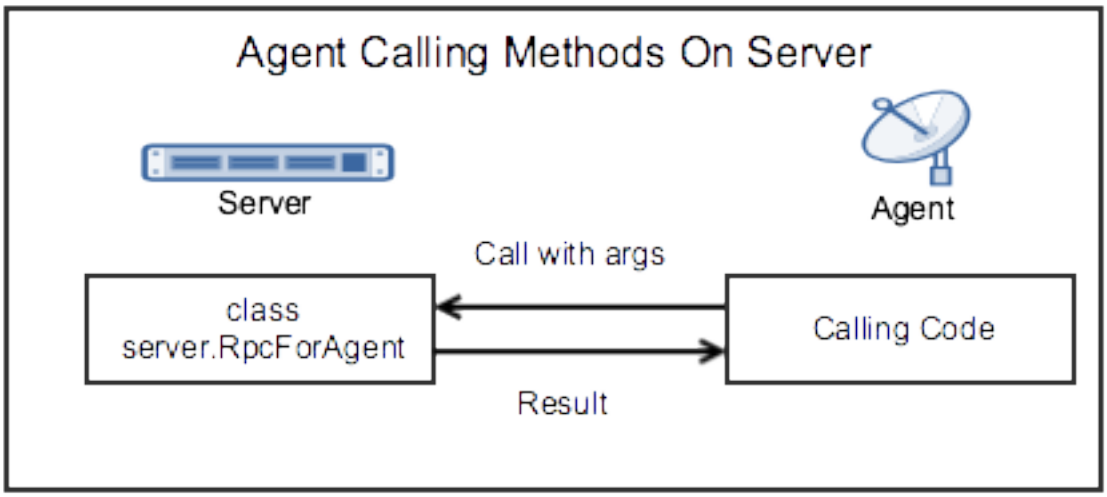
Note: Vortex RPC calls return `twisted.internet.defer.Deferred`, regardless of what the actual method returns.



Server RPC Setup

In this section we setup the files required to define an RPC on the server that will only accept calls from the agent.

The RPC example could be much simpler, the intenftion 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/server/agent_handlers` package, with the commands

```
mkdir peek_plugin_tutorial/_private/server/agent_handlers
touch peek_plugin_tutorial/_private/server/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 it will be good practice to have multiple files if the require RPC methods grow too large.

Create the file `peek_plugin_tutorial/_private/server/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.server.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 it's self 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 server 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 twisteds 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/server/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 `ServerEntryHook.py`

We need to update `ServerEntryHook.py`, to initialise the RpcForAgent.

Edit the file `peek_plugin_tutorial/_private/server/ServerEntryHook.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 Server RPC

This section implements the code in the agent that will call the RPC methods that the server has defined.

Add File `AgentToServerRpcCallExample.py`

File `AgentToServerRpcCallExample.py` defines the methods the agent will call via RPC.

In this example we have just one file, however it 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/AgentToServerRpcCallExample.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.server.agent_handlers.RpcForAgent import␣
↪RpcForAgent
from peek_plugin_tutorial._private.storage.StringIntTuple import StringIntTuple

logger = logging.getLogger(__name__)


class AgentToServerRpcCallExample:
    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 AgentToServerRpcCallExample.

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 .AgentToServerRpcCallExample import AgentToServerRpcCallExample
```

2. Add this line just before the `logger.debug("Started")` line at the end of the `start()` method:

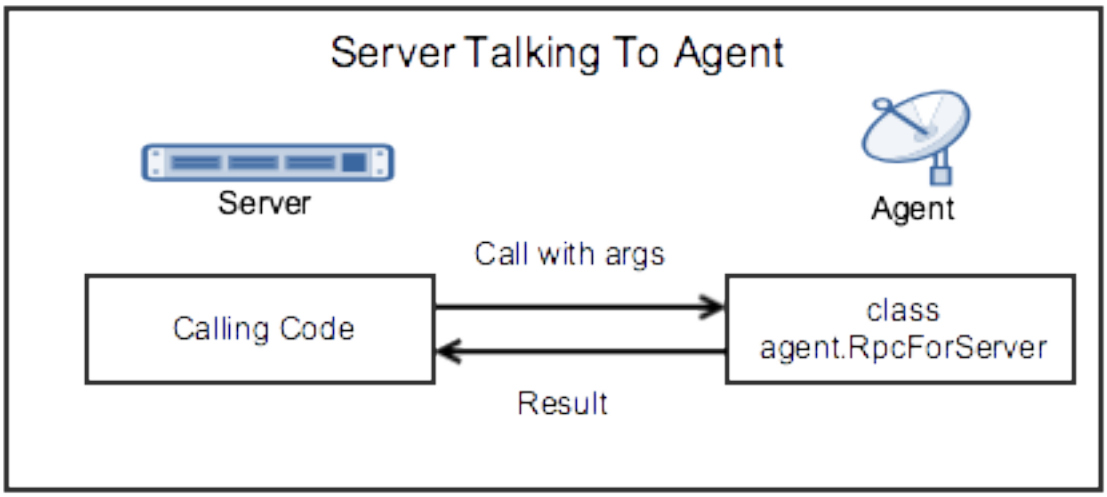
```
# Initialise and start the AgentToServerRpcCallExample
self._loadedObjects.append(AgentToServerRpcCallExample().start())
```

The agent will now call the server RPC methods.

Agent RPC Setup

In this section we setup the files required to define an RPC on the agent that the server 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 `RpcForServer.py`

File `RpcForServer.py` defines the methods the server will call via RPC.

Create the file `peek_plugin_tutorial/_private/agent/RpcForServer.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 RpcForServer:
    def __init__(self):
        pass

    def makeHandlers(self):
        """ Make Handlers

        In this method we start all the RPC handlers
        start() returns an instance of it's self so we can simply yield the result
        of the start method.

        """

        yield self.subInts.start(funcSelf=self)
        logger.debug("Server 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 RpcForServer.

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 .RpcForServer import RpcForServer
   ```

2. Add this line just before the `logger.debug("Started")` line at the end of the `start()` method:

   ```
   # Initialise and start the RPC for Server
   self._loadedObjects.extend(RpcForServer().makeHandlers())
   ```

The sever side RPC is now setup.

Server Calling Agent RPC

This section implements the code in the server that will call the RPC methods that the agent has defined.

Add File `ServerToAgentRpcCallExample.py`

File `ServerToAgentRpcCallExample.py` defines the methods the server will call via RPC.

Create the file `peek_plugin_tutorial/_private/server/ServerToAgentRpcCallExample.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.RpcForServer import RpcForServer

logger = logging.getLogger(__name__)


class ServerToAgentRpcCallExample:
    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 RpcForServer.subInts(7, kwval1=5)
        logger.debug("seedInt result = %s (Should be 2)", result)

    def shutdown(self):
        pass
```

Edit File `ServerEntryHook.py`

We need to update `ServerEntryHook.py`, to initialise the ServerToAgentRpcCallExample.

Edit the file `peek_plugin_tutorial/_private/server/ServerEntryHook.py`:

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

   ```
   from .ServerToAgentRpcCallExample import ServerToAgentRpcCallExample
   ```

2. Add this line just before the `logger.debug("Started")` line at the end of the `start()` method:

   ```
   # Initialise and start the RPC for Server
   self._loadedObjects.append(ServerToAgentRpcCallExample().start())
   ```

The server will now call the RPC method on the agent when it starts.

Testing

1. Open a command window and run: `run_peek_server`
2. Open a command window and run: `run_peek_agent`
3. Examine the logs of both command windows

`run_peek_server` log example:

```
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↪private.server.agent_handlers.RpcForAgent.RpcForAgent.updateStatus
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.server.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.server.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↪private.server.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↪private.server.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↪private.server.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↪private.server.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↪private.server.agent_handlers.RpcForAgent.RpcForAgent.addStringInt
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC call for peek_plugin_tutorial._
↪private.server.agent_handlers.RpcForAgent.RpcForAgent.updateStatus
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.server.controller.
↪MainController:Agent said : Agent RPC Example Completed
```

`run_peek_agent` log example:

```
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↪private.server.agent_handlers.RpcForAgent.RpcForAgent.updateStatus
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↪tutorial._private.server.agent_handlers.RpcForAgent.RpcForAgent.updateStatus
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.agent.
↪AgentToServerRpcCallExample:seedInt = 5
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↪private.server.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↪tutorial._private.server.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.agent.
↪AgentToServerRpcCallExample:seedInt = 12
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↪private.server.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↪tutorial._private.server.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.agent.
↪AgentToServerRpcCallExample:seedInt = 19
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↪private.server.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↪tutorial._private.server.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.agent.
↪AgentToServerRpcCallExample:seedInt = 26
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↪private.server.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.server.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.agent.
↪AgentToServerRpcCallExample:seedInt = 33
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↪private.server.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↪tutorial._private.server.agent_handlers.RpcForAgent.RpcForAgent.addInts
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.agent.
↪AgentToServerRpcCallExample:seedInt = 40
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↪private.server.agent_handlers.RpcForAgent.RpcForAgent.addStringInt
19-Apr-2017 09:24:42 DEBUG peek_plugin_tutorial._private.agent.
↪AgentToServerRpcCallExample:runWithInlineCallback finished
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↪tutorial._private.server.agent_handlers.RpcForAgent.RpcForAgent.addStringInt
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Calling RPC for peek_plugin_tutorial._
↪private.server.agent_handlers.RpcForAgent.RpcForAgent.updateStatus
19-Apr-2017 09:24:42 DEBUG vortex.rpc.RPC:Received RPC result for peek_plugin_
↪tutorial._private.server.agent_handlers.RpcForAgent.RpcForAgent.updateStatus
```

8.9.16 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.

Anther 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 Server service can only use the API published by the code in `peek_plugin_two` that runs on the Server service.

What are APIs

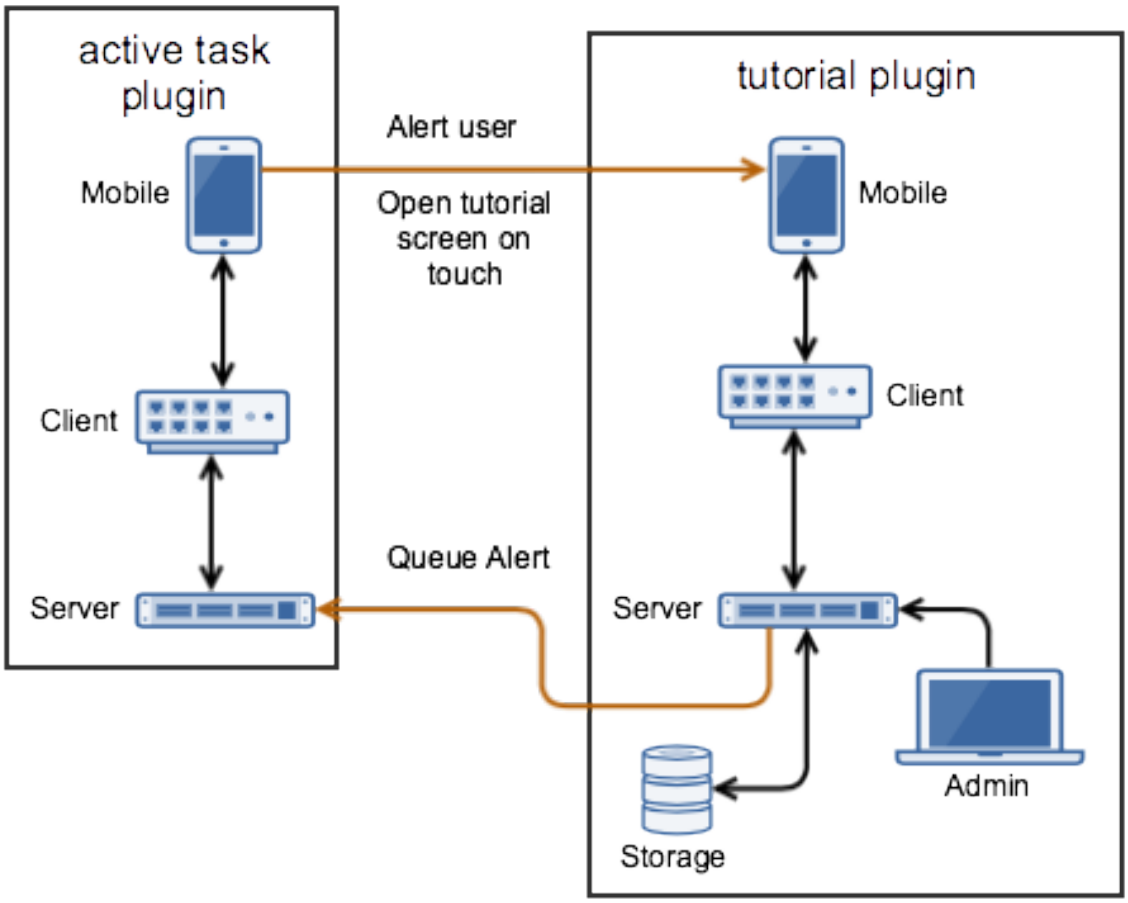
An API is an application programming interface, in python side Peek terms, it's an exposed abstract class or classes that other plugins can import and use. By "exposed" we mean, anything not under the "_private" package.

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 Server service in the `peek_plugin_tutorial` plugin.

We'll then get the API for the `peek_plugin_active_task` plugin and create a task.



Setup Server API

In this section, we define an API on the Peek Server 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
```

(continues on next page)

(continued from previous page)

```
    This tuple is publicly exposed and will be the result of the doSomething api call.
    """
    __tupleType__ = tutorialTuplePrefix + 'DoSomethingTuple'

    #:  The result of the doSomething
    result = TupleField(defaultValue=dict)
```

Add Package `server`

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.server` python package will contain the exposed API abstract classes.

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

```
mkdir peek_plugin_tutorial/server
touch peek_plugin_tutorial/server/__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/server/TutorialApi.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 insance of this class will be passed to other APIs when they ask for it.

Create the file `peek_plugin_tutorial/_private/server/TutorialApi.py` and populate it with the following contents.

```
from peek_plugin_tutorial._private.server.controller.MainController import␣
↪MainController
from peek_plugin_tutorial.server.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 `ServerEntryHook.py`

We need to update `ServerEntryHook.py`, to initialise the API object.

Edit the file `peek_plugin_tutorial/_private/server/ServerEntryHook.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 `ServerEntryHook` class:

```
@property
def publishedServerApi(self) -> object:
    """ Published Server 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 Server API

In this section we'll get a reference to the Active Task API and then create a task on the mobile UI.

Note: In order to use this example, you will need to have the `peek_core_user` plugin installed and enabled in both the Client and Server 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_active_task` plugin installed and enabled in both the Client and Server services, via their config.json files.

The active task plugin is public, it can be installed with **`pip install peek-plugin-active-task`**.

Add File `ExampleUseTaskApi.py`

File `ExampleUseTaskApi.py` contains the code that uses the Active Tasks API.

Create the file `peek_plugin_tutorial/_private/server/ExampleUseTaskApi.py` and populate it with the following contents.

Replace the `"userId"` with your user id.

```
import logging
from datetime import datetime

from twisted.internet import reactor
from twisted.internet.defer import inlineCallbacks

from peek_plugin_active_task.server.ActiveTaskApiABC import ActiveTaskApiABC, NewTask
from peek_plugin_tutorial._private.server.controller.MainController import
↪MainController
from peek_plugin_tutorial._private.PluginNames import tutorialPluginName
```

(continues on next page)

(continued from previous page)

```
logger = logging.getLogger(__name__)


class ExampleUseTaskApi:
    def __init__(self, mainController: MainController, activeTaskApi:␣
↪ActiveTaskApiABC):
        self._mainController = mainController
        self._activeTaskApi = activeTaskApi

    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.utcnow()),
            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 active tasks API
        yield self._activeTaskApi.addTask(newTask)

        logger.debug("Task Sent")

    def shutdown(self):
        pass
```

Edit File `ServerEntryHook.py`

We need to update `ServerEntryHook.py`, to initialise the example code

Edit the file `peek_plugin_tutorial/_private/server/ServerEntryHook.py`:

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

```
from peek_plugin_active_task.server.ActiveTaskApiABC import ActiveTaskApiABC
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 Active Task
activeTaskApi = self.platform.getOtherPluginApi("peek_plugin_active_task")
assert isinstance(activeTaskApi, ActiveTaskApiABC), "Wrong activeTaskApi"
```

(continues on next page)

(continued from previous page)

```
# Initialise the example code that will send the test task
self._loadedObjects.append(
        ExampleUseTaskApi(mainController, activeTaskApi).start()
)
```

Testing

1. Open mobile Peek web app
2. Tap Task icon located in the top right corner
3. You will see the task in the list

8.9.17 Add Plugin TypeScript APIs (TODO)

Overview

This document will describe how to use the APIs between plugins running on the:

- Mobile
- Desktop
- 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.

8.9.18 Use Plugin APIs (TODO)

Overview

This doc describes how to retrieve and use APIs from other plugins.

8.9.19 Add Worker Service

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

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/WorkerEntryHook.py` and populate it with the following contents.

```
import logging

from peek_plugin_base.worker.PluginWorkerEntryHookABC import PluginWorkerEntryHookABC

logger = logging.getLogger(__name__)


class WorkerEntryHook(PluginWorkerEntryHookABC):
    def __init__(self, *args, **kwargs):
        """ Constructor """
        # Call the base classes constructor
        PluginWorkerEntryHookABC.__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
```

(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")
```

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"
    ],
```

(continues on next page)

(continued from previous page)

```
    "worker": {
    }
}
```

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

Running on the Worker Service

Edit `~/peek-worker.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
...
DEBUG peek_plugin_tutorial._private.worker.WorkerEntryHook:Loaded
DEBUG peek_plugin_tutorial._private.worker.WorkerEntryHook:Started
...
```

8.10 What Next?

Refer back to the *How to Use Peek Documentation* guide to see which document to follow next.

CHAPTER 9

Troubleshooting

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
# Expcect 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
# Expcect 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-ns, 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
# Expcect 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

CHAPTER 11

API Reference

11.1 File `plugin_package.json`

This page will describe the options in the `plugin_package.json`

11.2 Synerty Peek

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*

(M) PluginAgentEntryHookABC

class peek_plugin_base.agent.PluginAgentEntryHookABC.**PluginAgentEntryHookABC**(*pluginName: str*, *pluginRootDir: str*, *platform: 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.PeekPlatformMobileHttpHookABC.PeekPlatformMobileHttpHookABC*, *peek_plugin_base.client.PeekPlatformDesktopHttpHookABC.PeekPlatformDesktopHttpHookABC*, *peek_plugin_base.PeekPlatformServerInfoHookABC.PeekPlatformServerInfoHookABC*, *peek_plugin_base.PeekPlatformFileStorageHookABC.PeekPlatformFileStorageHookABC*

(M) PeekPlatformDesktopHttpHookABC

class peek_plugin_base.client.PeekPlatformDesktopHttpHookABC.**PeekPlatformDesktopHttpHookABC**

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) beween the server and the agent, worker and client.

addDesktopResource(*pluginSubPath: bytes*, *resource: twisted.web.resource.IResource*) → 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

addDesktopStaticResourceDir(*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

rootDesktopResource
Site Root Resource

This returns the root site resource for this plugin.

(M) PeekPlatformMobileHttpHookABC

class peek_plugin_base.client.PeekPlatformMobileHttpHookABC.**PeekPlatformMobileHttpHookABC**
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) beween the server and the agent, worker and client.

addMobileResource(*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

addMobileStaticResourceDir(*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

rootMobileResource
Site Root Resource

This returns the root site resource for this plugin.

(M) PluginClientEntryHookABC

class peek_plugin_base.client.PluginClientEntryHookABC.**PluginClientEntryHookABC**(*pluginName: str*, *pluginRootDir: str*, *platform: peek_plugin_ba*

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) beween 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) beween the server and the agent, worker and client.

These resources will not be availible 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*

dbConnectString
DB Connect String

Returns The SQLAlchemy database engine connection string/url.

(M) PluginServerEntryHookABC

class peek_plugin_base.server.PluginServerEntryHookABC.**PluginServerEntryHookABC**(*pluginName: str*, *pluginRootDir: str*, *platform: peek_plugin_ba*

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.**PluginServerStorageEntryHook**
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.

```
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

celeryApp
Celery App

This plugin property is called by the platform when the server is initialising the plugin.

The instance of the celery app returned by this property will be configured, allowing tasks linked to it to work with the Peek celerty task queues.

Returns An instance of C{celery.app.base.Celery}

```
from peek_plugin_noop.worker.NoopCeleryApp import celeryApp
return celeryApp
```

(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*, *enableForeignKeys=False*, *enableCreateAll=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 ormSesson 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) StorageUtil

peek_plugin_base.storage.StorageUtil.**makeCoreValuesSubqueryCondition**(*engine, column, 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

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) 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*, *pluginRootDir: str*, *platform: peek_plugin_ba*

Bases: *peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC*

celeryApp
Celery App

Return the workers instance of the celery app.

This will be configured with the platforms celery setup.

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'**
The vortex name for the Admin browser client

peek_plugin_base.PeekVortexUtil.**peekAgentName = 'peek-agent'**
The vortex name for the Agent service

peek_plugin_base.PeekVortexUtil.**peekClientName = 'peek-client'**
The vortex name for the Client service

peek_plugin_base.PeekVortexUtil.**peekDesktopName = 'peek-desktop'**
The vortex name for the Desktop browser clients

peek_plugin_base.PeekVortexUtil.**peekMobileName = 'peek-mobile'**
The vortex name for the Mobile device/browser clients

peek_plugin_base.PeekVortexUtil.**peekServerName = 'peek-server'**
The vortex name for the Server service

peek_plugin_base.PeekVortexUtil.**peekWorkerName = 'peek-worker'**
The vortex name for the Worker service

(M) PluginCommonEntryHookABC

class peek_plugin_base.PluginCommonEntryHookABC.**PluginCommonEntryHookABC**(*pluginName: str*, *pluginRootDir: str*)

Bases: object

load() → None
Load

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

CHAPTER 12

Upgrade Notes

12.1 v2.0.x Upgrade Notes

12.1.1 Changes

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.

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 resurected. 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.

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.

Platform Backend

The Peek Admin UI now requires a login.

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

Windows Services

Nil, carry on.

12.1.2 Linux Deployment

Nil

12.1.3 macOS Deployment

Nil

12.1.4 iOS Deployment

Peek v2.0.x does not have support for iOS, this will be updated in a figure release

12.1.5 Windows Deployment

Nil.

Note: The windows deployment will change to use Windows Subsystem for Linux in a future release.

12.1.6 Enable New Plugins

Update the peek config.json files.

1. Edit each of C:Userspeekpeek-XXXX.homeconfig.json
2. Add `peek_plugin_diagram_postioner` just after `peek_plugin_diagram_trace`
3. **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
```

Start up the Peek Server service, it will rebuild the admin site.

Restart all Peek services.

For windows, restart the `peek-server` service then start the `peek-restarter` service, the agent, worker and client will now start.

12.1.7 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 / PoF 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 / PoF 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_server 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

12.2 v1.3.x Upgrade Notes

12.2.1 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.

12.2.2 Linux Deployment

Nil

12.2.3 macOS Deployment

Nil

12.2.4 iOS Deployment

Peek v1.3.x does not have support for iOS, this will be updated in a figure release

12.2.5 Windows Deployment

Nil.

Note: The windows deployment will change to use Windows Subsystem for Linux in a future release.

12.2.6 Enable New Plugins

Update the peek config.json files.

1. Edit each of C:Userspeekpeek-XXXX.homeconfig.json
2. Add `peek_plugin_branch` to the start
3. Add `peek_plugin_pof_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-server` service then start the `peek-restarter` service, the agent, worker and client will now start.

12.3 v1.2.x Upgrade Notes

12.3.1 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.

PoF Connectivity Model Loader

The PoF 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.

12.3.2 Linux Deployment

The Linux service scripts have been modified to use systemd, This is supported by both RHEL7 and Debian9. This chane 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_server peek_worker peek_agent peek_client
do
    service ${service} stop
    update-rc.d ${service} disable
    rm /etc/init.d/${service}
done
```

12.3.3 macOS Deployment

Update to the latest XCode, 10.1.

12.3.4 iOS Deployment

The Peek Mobile native app now supports iOS 12.1.

12.3.5 Windows Deployment

nil.

12.3.6 Enable New Plugins

Update the peek config.json files.

1. Edit each of C:Userspeekpeek-XXXX.homeconfig.json
2. Add `peek_plugin_graphdb` after `peek_plugin_livedb`
3. Add `peek_plugin_pof_graphdb_loader` after `peek_plugin_pof_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 -> PoF 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-server` service then start the `peek-restarter` service, the agent, worker and client will now start.

12.4 v1.1.0 Upgrade Notes

12.4.1 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 an 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 reactivly observed from the peek client.

There are preformance 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

12.4.2 Linux Deployment

Nil, carry on being awesome.

12.4.3 macOS Deployment

Update to the latest XCode, 9.4.

12.4.4 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 PostgresSQL

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

Deploy the plugins.

Deploy Peek Plugins

12.4.5 Enable New Plugins

Update the peek config.json files.

1. Edit each of C:Userspeekpeek-XXXX.homeconfig.json
2. Add `peek_plugin_docdb` after `peek_plugin_livedb`
3. Add `peek_core_search` after `peek_plugin_livedb`
4. Add `peek_plugin_pof_equipment_loader` after `peek_plugin_pof_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 -> PoF 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-server` service then start the `peek-restarter` service, the agent, worker and client will now start.

12.5 v0.10.0 Upgrade Notes

12.5.1 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 respectivly.

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_client 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 setttings change

Windows Services

Peek v0.10.0 now contains windows services. These release notes will describe how to install the services.

12.5.2 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 Platform

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.

Deploy the plugins.

Deploy Peek Plugins

12.6 v0.6.0 Upgrade Notes

The following modifications are required to upgrade plugins to run on the v0.6.0 version of the platform

12.6.1 NPM : peek-mobile-util

The `peek-mobile-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-mobile-util/peek-
↪util/g' {} \;
```

The `peek-mobile-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

Python Module Index

p

peek_plugin_base, 271
peek_plugin_base.agent, 271
peek_plugin_base.agent.PeekAgentPlatformHookABC, 271
peek_plugin_base.agent.PluginAgentEntryHookABC, 272
peek_plugin_base.client, 272
peek_plugin_base.client.PeekClientPlatformHookABC, 272
peek_plugin_base.client.PeekPlatformDesktopHttpHookABC, 272
peek_plugin_base.client.PeekPlatformMobileHttpHookABC, 273
peek_plugin_base.client.PluginClientEntryHookABC, 274
peek_plugin_base.PeekPlatformCommonHookABC, 281
peek_plugin_base.PeekPlatformFileStorageHookABC, 282
peek_plugin_base.PeekPlatformServerInfoHookABC, 282
peek_plugin_base.PeekVortexUtil, 282
peek_plugin_base.PluginCommonEntryHookABC, 283
peek_plugin_base.PluginPackageFileConfig, 283
peek_plugin_base.server, 274
peek_plugin_base.server.PeekPlatformAdminHttpHookABC, 274
peek_plugin_base.server.PeekPlatformServerHttpHookABC, 275
peek_plugin_base.server.PeekServerPlatformHookABC, 275
peek_plugin_base.server.PluginServerEntryHookABC, 276
peek_plugin_base.server.PluginServerStorageEntryHookABC, 276
peek_plugin_base.server.PluginServerWorkerEntryHookABC, 277
peek_plugin_base.storage, 277
peek_plugin_base.storage.AlembicEnvBase, 277
peek_plugin_base.storage.DbConnection, 278
peek_plugin_base.storage.StorageUtil, 279
peek_plugin_base.storage.TypeDecorators, 279
peek_plugin_base.worker, 280
peek_plugin_base.worker.CeleryDbConn, 280
peek_plugin_base.worker.CeleryDbConnInit, 280
peek_plugin_base.worker.PeekWorkerPlatformHookABC, 281
peek_plugin_base.worker.PluginWorkerEntryHookABC, 281

Index

A

addAdminResource() (*peek_plugin_base.server.PeekPlatformAdminHttpHookABC.PeekPlatformAdminHttpHookABC method*), 274

addAdminStaticResourceDir() (*peek_plugin_base.server.PeekPlatformAdminHttpHookABC.PeekPlatformAdminHttpHookABC method*), 274

addDesktopResource() (*peek_plugin_base.client.PeekPlatformDesktopHttpHookABC.PeekPlatformDesktopHttpHookABC method*), 272

addDesktopStaticResourceDir() (*peek_plugin_base.client.PeekPlatformDesktopHttpHookABC.PeekPlatformDesktopHttpHookABC method*), 272

addMobileResource() (*peek_plugin_base.client.PeekPlatformMobileHttpHookABC.PeekPlatformMobileHttpHookABC method*), 273

addMobileStaticResourceDir() (*peek_plugin_base.client.PeekPlatformMobileHttpHookABC.PeekPlatformMobileHttpHookABC method*), 273

addServerResource() (*peek_plugin_base.server.PeekPlatformServerHttpHookABC.PeekPlatformServerHttpHookABC method*), 275

addServerStaticResourceDir() (*peek_plugin_base.server.PeekPlatformServerHttpHookABC.PeekPlatformServerHttpHookABC method*), 275

AlembicEnvBase (*class in peek_plugin_base.storage.AlembicEnvBase*), 277

angularFrontendAppDir (*peek_plugin_base.client.PluginClientEntryHookABC.PluginClientEntryHookABC attribute*), 274

angularMainModule (*peek_plugin_base.client.PluginClientEntryHookABC.PluginClientEntryHookABC attribute*), 274

B

bind_expression() (*peek_plugin_base.storage.TypeDecorators.PeekLargeBinary method*), 279

C

celeryApp (*peek_plugin_base.server.PluginServerWorkerEntryHookABC.PluginServerWorkerEntryHookABC attribute*), 277

celeryApp (*peek_plugin_base.worker.PluginWorkerEntryHookABC.PluginWorkerEntryHookABC attribute*), 281

celeryApp (*peek_plugin_base.worker.PluginWorkerEntryHookABC.PluginWorkerEntryHookABC attribute*), 281

checkSchemaExists() (*peek_plugin_base.storage.DbConnection.DbConnection method*), 278

checkSchemaExists() (*peek_plugin_base.storage.DbConnection.DbConnection method*), 278

checksum (*peek_plugin_base.storage.PluginPackageFileConfig.PluginPackageFileConfig attribute*), 284

convertToCoreSqlaInsert() (*in module peek_plugin_base.storage.DbConnection*), 279

D

DbConnection (*class in peek_plugin_base.storage.DbConnection*), 278

dbConnectString (*peek_plugin_base.server.PeekServerPlatformHookABC.PeekServerPlatformHookABC attribute*), 275

dbEngine (*peek_plugin_base.server.PluginServerStorageEntryHookABC.PluginServerStorageEntryHookABC attribute*), 276

dbEngine (*peek_plugin_base.storage.DbConnection.DbConnection attribute*), 278

dbMetadata (*peek_plugin_base.server.PluginServerStorageEntryHookABC.PluginServerStorageEntryHookABC attribute*), 276

dbSessionCreator (*peek_plugin_base.server.PluginServerEntryHookABC.PluginServerEntryHookABC attribute*), 276

dbSessionCreator (*peek_plugin_base.server.PluginServerStorageEntryHookABC.PluginServerStorageEntryHookABC attribute*), 277

E

ensureSchemaExists() (*in module*

peek_plugin_base.storage.AlembicEnvBase), 277

F

fileStorageDirectory (*peek_plugin_base.PeekPlatformFileStorageHookABC.PeekPlatformFileStorageHookABC attribute*), 282

G

getDbEngine() (*in module peek_plugin_base.worker.CeleryDbConn*), 280
getDbSession() (*in module peek_plugin_base.worker.CeleryDbConn*), 280
getOtherPluginApi() (*peek_plugin_base.PeekPlatformCommonHookABC.PeekPlatformCommonHookABC method*), 281

I

impl (*peek_plugin_base.storage.TypeDecorators.PeekLargeBinary attribute*), 280
initWorkerConnString() (*in module peek_plugin_base.worker.CeleryDbConnInit*), 280
initWorkerProcessDbConn() (*in module peek_plugin_base.worker.CeleryDbConnInit*), 280
isMssqlDialect() (*in module peek_plugin_base.storage.AlembicEnvBase*), 277
isPostGreSQLDialect() (*in module peek_plugin_base.storage.AlembicEnvBase*), 277

L

load() (*peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC method*), 283

M

makeCoreValuesSubqueryCondition() (*in module peek_plugin_base.storage.StorageUtil*), 279
makeOrmValuesSubqueryCondition() (*in module peek_plugin_base.storage.StorageUtil*), 279
migrate() (*peek_plugin_base.storage.DbConnection.DbConnection method*), 278
migrateStorageSchema() (*peek_plugin_base.server.PluginServerEntryHookABC.PluginServerEntryHookABC method*), 276

N

name (*peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC attribute*), 283

O

ormSessionCreator (*peek_plugin_base.storage.DbConnection.DbConnection attribute*), 278

P

packageCfg (*peek_plugin_base.PluginCommonEntryHookABC.PluginC attribute*), 283
peek_plugin_base (*module*), 271
peek_plugin_base.agent (*module*), 271
peek_plugin_base.agent.PeekAgentPlatformHookABC (*module*), 271
peek_plugin_base.agent.PluginAgentEntryHookABC (*module*), 272
peek_plugin_base.client (*module*), 272
peek_plugin_base.client.PeekClientPlatformHookABC (*module*), 272
peek_plugin_base.client.PeekPlatformDesktopHttpHook (*module*), 272
peek_plugin_base.client.PeekPlatformMobileHttpHookA (*module*), 273
peek_plugin_base.client.PluginClientEntryHookABC (*module*), 274
peek_plugin_base.PeekPlatformCommonHookABC (*module*), 281
peek_plugin_base.PeekPlatformFileStorageHookABC (*module*), 282
peek_plugin_base.PeekPlatformServerInfoHookABC (*module*), 282
peek_plugin_base.PeekVortexUtil (*module*), 282
peek_plugin_base.PluginCommonEntryHookABC (*module*), 283
peek_plugin_base.PluginPackageFileConfig (*module*), 283
peek_plugin_base.server (*module*), 274
peek_plugin_base.server.PeekPlatformAdminHttpHookA (*module*), 274
peek_plugin_base.server.PeekPlatformServerHttpHookA (*module*), 275
peek_plugin_base.server.PeekServerPlatformHookABC (*module*), 275
peek_plugin_base.server.PluginServerEntryHookABC (*module*), 276
peek_plugin_base.server.PluginServerStorageEntryHoo (*module*), 276
peek_plugin_base.server.PluginServerWorkerEntryHook (*module*), 277
peek_plugin_base.storage (*module*), 277
peek_plugin_base.storage.AlembicEnvBase (*module*), 277
peek_plugin_base.storage.DbConnection (*module*), 278

peek_plugin_base.storage.StorageUtil (*module*), 279
peek_plugin_base.storage.TypeDecorators (*module*), 279
peek_plugin_base.worker (*module*), 280
peek_plugin_base.worker.CeleryDbConn (*module*), 280
peek_plugin_base.worker.CeleryDbConnInit (*module*), 280
peek_plugin_base.worker.PeekWorkerPlatformHookABC (*module*), 281
peek_plugin_base.worker.PluginWorkerEntryHookABC (*module*), 281
peekAdminName (*in module peek_plugin_base.PeekVortexUtil*), 282
peekAgentName (*in module peek_plugin_base.PeekVortexUtil*), 282
PeekAgentPlatformHookABC (*class in peek_plugin_base.agent.PeekAgentPlatformHookABC*), 271
peekClientName (*in module peek_plugin_base.PeekVortexUtil*), 282
PeekClientPlatformHookABC (*class in peek_plugin_base.client.PeekClientPlatformHookABC*), 272
peekDesktopName (*in module peek_plugin_base.PeekVortexUtil*), 282
PeekLargeBinary (*class in peek_plugin_base.storage.TypeDecorators*), 279
peekMobileName (*in module peek_plugin_base.PeekVortexUtil*), 282
PeekPlatformAdminHttpHookABC (*class in peek_plugin_base.server.PeekPlatformAdminHttpHookABC*), 274
PeekPlatformCommonHookABC (*class in peek_plugin_base.PeekPlatformCommonHookABC*), 281
PeekPlatformDesktopHttpHookABC (*class in peek_plugin_base.client.PeekPlatformDesktopHttpHookABC*), 272
PeekPlatformFileStorageHookABC (*class in peek_plugin_base.PeekPlatformFileStorageHookABC*), 282
PeekPlatformMobileHttpHookABC (*class in peek_plugin_base.client.PeekPlatformMobileHttpHookABC*), 273
PeekPlatformServerHttpHookABC (*class in peek_plugin_base.server.PeekPlatformServerHttpHookABC*), 275
PeekPlatformServerInfoHookABC (*class in peek_plugin_base.PeekPlatformServerInfoHookABC*), 282
peekServerHost (*peek_plugin_base.PeekPlatformServerInfoHookABC.PeekPlatformServerInfoHookABC attribute*), 282
peekServerHttpPort (*peek_plugin_base.PeekPlatformServerInfoHookABC.PeekPlatformServerInfoHookABC attribute*), 282
peekServerName (*in module peek_plugin_base.PeekVortexUtil*), 282
PeekServerPlatformHookABC (*class in peek_plugin_base.server.PeekServerPlatformHookABC*), 275
peekWorkerName (*in module peek_plugin_base.PeekVortexUtil*), 282
PeekWorkerPlatformHookABC (*class in peek_plugin_base.worker.PeekWorkerPlatformHookABC*), 281
pgCopyInsert() (*in module peek_plugin_base.storage.DbConnection*), 279
platform (*peek_plugin_base.agent.PluginAgentEntryHookABC.PluginAgentEntryHookABC attribute*), 272
platform (*peek_plugin_base.client.PluginClientEntryHookABC.PluginClientEntryHookABC attribute*), 274
platform (*peek_plugin_base.server.PluginServerEntryHookABC.PluginServerEntryHookABC attribute*), 276
platform (*peek_plugin_base.worker.PluginWorkerEntryHookABC.PluginWorkerEntryHookABC attribute*), 281
PluginAgentEntryHookABC (*class in peek_plugin_base.agent.PluginAgentEntryHookABC*), 272
PluginClientEntryHookABC (*class in peek_plugin_base.client.PluginClientEntryHookABC*), 274
PluginCommonEntryHookABC (*class in peek_plugin_base.PluginCommonEntryHookABC*), 283
PluginPackageFileConfig (*class in peek_plugin_base.PluginPackageFileConfig*), 283
PluginServerEntryHookABC (*class in peek_plugin_base.server.PluginServerEntryHookABC*), 276
PluginServerStorageEntryHookABC (*class in peek_plugin_base.server.PluginServerStorageEntryHookABC*), 276
PluginServerWorkerEntryHookABC (*class in peek_plugin_base.server.PluginServerWorkerEntryHookABC*), 277
PluginWorkerEntryHookABC (*class in peek_plugin_base.worker.PluginWorkerEntryHookABC*), 281
prefetchDeclarativeIds() (*in module peek_plugin_base.worker.CeleryDbConn*), 280
prefetchDeclarativeIds() (*peek_plugin_base.server.PluginServerStorageEntryHookABC.PluginServerStorageEntryHookABC*)

method), 277
prefetchDeclarativeIds() (*peek_plugin_base.storage.DbConnection.DbConnection method*), 279
publishedAgentApi (*peek_plugin_base.agent.PluginAgentEntryHookABC.PluginAgentEntryHookABC attribute*), 272
publishedClientApi (*peek_plugin_base.client.PluginClientEntryHookABC.PluginClientEntryHookABC attribute*), 274
publishedServerApi (*peek_plugin_base.server.PluginServerEntryHookABC.PluginServerEntryHookABC attribute*), 276

R

rootAdminResource (*peek_plugin_base.server.PeekPlatformAdminHttpHookABC.PeekPlatformAdminHttpHookABC attribute*), 275
rootDesktopResource (*peek_plugin_base.client.PeekPlatformDesktopHttpHookABC.PeekPlatformDesktopHttpHookABC attribute*), 273
rootDir (*peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC attribute*), 283
rootMobileResource (*peek_plugin_base.client.PeekPlatformMobileHttpHookABC.PeekPlatformMobileHttpHookABC attribute*), 273
rootServerResource (*peek_plugin_base.server.PeekPlatformServerHttpHookABC.PeekPlatformServerHttpHookABC attribute*), 275
run() (*peek_plugin_base.storage.AlembicEnvBase.AlembicEnvBase method*), 277

S

serviceId (*peek_plugin_base.PeekPlatformCommonHookABC.PeekPlatformCommonHookABC attribute*), 281
setConnStringForWindows() (*in module peek_plugin_base.worker.CeleryDbConn*), 280
shutdownWorkerProcessDbConn() (*in module peek_plugin_base.worker.CeleryDbConnInit*), 280
start() (*peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC method*), 283
stop() (*peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC method*), 283

T

taskEndCloseSession() (*in module peek_plugin_base.worker.CeleryDbConnInit*), 281
title (*peek_plugin_base.PluginCommonEntryHookABC.PluginCommonEntryHookABC attribute*), 283

U

unload() (*peek_plugin_base.PluginCommonEntryHookABC.PluginCom method*), 283