



2015-06-01

Tethys Platform: A Development and Hosting Platform for Water Resources Web Apps

Nathan R. Swain

Brigham Young University - Provo

Follow this and additional works at: <https://scholarsarchive.byu.edu/etd>

 Part of the [Civil and Environmental Engineering Commons](#)

BYU ScholarsArchive Citation

Swain, Nathan R., "Tethys Platform: A Development and Hosting Platform for Water Resources Web Apps" (2015). *All Theses and Dissertations*. 5832.

<https://scholarsarchive.byu.edu/etd/5832>

This Dissertation is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

Tethys Platform: A Development and Hosting
Platform for Water Resources Web Apps

Nathan R. Swain

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

E. James Nelson, Chair
Norman L. Jones
Daniel P. Ames
Gustavious P. Williams
A. Woodruff Miller

Department of Civil and Environmental Engineering

Brigham Young University

June 2015

Copyright © 2015 Nathan R. Swain

All Rights Reserved

ABSTRACT

Tethys Platform: A Development and Hosting Platform for Water Resources Web Apps

Nathan R. Swain

Department of Civil and Environmental Engineering, BYU
Doctor of Philosophy

The interactive nature of web applications or “web apps” makes it an excellent medium for conveying complex scientific concepts to lay audiences and creating decision support tools that harness cutting edge modeling techniques. However, the technical expertise required to develop them represents a barrier for would-be developers. The barrier can be characterized by the following hurdles that developers must overcome: (1) identify, select, and install software that meet the spatial and computational capabilities commonly required for water resources modeling; (2) orchestrate the use of multiple FOSS and FOSS4G projects and navigate their differing application programming interfaces (APIs); (3) learn the multi-language programming skills required for modern web development; and (4) develop a web-safe and fully featured web site to host the app.

This research has resulted in two primary products that effectively lower the barrier to water resources web app development: (1) a literature review of free and open source software (i.e. software review) and (2) Tethys Platform. The software review included earth science web apps that were published in the peer-reviewed literature in the last decade and it was performed to determine which FOSS4G and FOSS web software has been used to develop such web apps. The review highlights 11 FOSS4G software projects and 9 FOSS projects for web development that were used to develop 45 earth sciences web apps, which constitutes a significantly reduced list of possible software projects that could be used to meet the needs of water resources web app development—greatly lowering the barrier for entry to water resources web development.

While the software review addresses the hurdle of identifying FOSS software to provide a web framework and spatial data capabilities for water resources web apps, there are still other hurdles that needed to be overcome to make development more viable. Tethys Platform was developed to address these other hurdles and streamline the development of water resources web apps. It includes (1) a suite of free and open source software that address the unique data and computational needs common to water resources web app development, (2) a Python software development kit for incorporating the functionality of each software element into web apps and streamlining their development, and (3) a customizable web portal that is used to deploy the completed web apps. Tethys Platform has been used to develop a broad array of web apps for water resources modeling and decision support.

Keywords: Tethys Platform, water resources, modeling, software development kit, web app

ACKNOWLEDGEMENTS

At the outset of my doctoral degree and my involvement with CI-WATER I could not have anticipated the experiences that I would have or the ways in which I would grow both as a professional and as an individual. Though I fully expected to develop bleeding-edge technologies, I did not see Tethys Platform on the horizon or the overwhelming interest and response to this work. I am overcome with gratitude for the many people who afforded me this opportunity.

I would like to thank my advisors, Jim Nelson and Norm Jones. They provided a compelling vision that inspired my efforts and they provided me the latitude to explore many different avenues that eventually led to Tethys Platform. I would also like to acknowledge Dan Ames for his guidance and especially for his recommendation to port the project from CKAN to Django—a decision that was critical for the longevity of the project. I would also like to thank Gus Williams and Wood Miller for their input and feedback as members of my committee.

I am indebted to Scott Christensen for his patience and invaluable advice. I would rarely implement a new feature or move forward with a plan until I had pitched the idea to Scott, because he always helped me root out the flaws and perfect the ideas. I would like to thank Jocelynn Anderson as the first Tethys app developer. Her work on the GSSHA Index Map Editor app was a driver for the development of many features. I am also grateful for Alan Snow and his work on the Streamflow Prediction Tool app, which put wind under the wings of Tethys Platform affording it national attention. I also want to recognize the efforts of the other early Tethys app developers on the CI-WATER and HydroShare teams. Tethys Platform would be nothing without the people who use it.

I also need to recognize my parents for instilling in me an attitude of open mindedness and the confidence to reach beyond myself. Lastly, I would be remiss if I didn't recognize my wife, Sunni. She has been a web developer much longer than I have, which was a large motivation behind my decision to embark on the CI-WATER project. In addition to countless acts of selfless love and support, she provided invaluable advice as I learned the ins and outs of web development. I would not have been able to complete or even participate in this project without her support.

This material is based upon work supported by the National Science Foundation under Grant No. 1135483.

TABLE OF CONTENTS

LIST OF TABLES	viii
LIST OF FIGURES	ix
1 Introduction.....	1
2 Software Review.....	6
2.1 Web GIS Software Review.....	7
2.1.1 Spatial Databases	8
2.1.2 Spatial Data Publishing.....	11
2.1.3 Mapping Libraries.....	13
2.1.4 Spatial Analysis	16
2.2 Web Development Software Review.....	18
2.2.1 Programming Languages	20
2.2.2 Web Frameworks	21
2.2.3 Content Management Systems.....	23
3 Tethys Platform.....	25
3.1 Tethys Software Suite.....	26
3.1.1 Spatial Database.....	27
3.1.2 Geoprocessing.....	27
3.1.3 Map Rendering.....	28
3.1.4 Visualization	28
3.1.5 Distributed Computing.....	29
3.1.6 File Dataset Storage	30
3.1.7 Docker Installation.....	30
3.2 Tethys Software Development Kit	31

3.2.1	App Development Framework.....	32
3.2.2	Command Line Interface	34
3.2.3	Template Gizmos API.....	35
3.2.4	Persistent Stores APIs	36
3.2.5	Dataset Services APIs	37
3.2.6	Web Processing Services API.....	37
3.2.7	Distributed Computing API	38
3.3	Tethys Portal	38
4	Applications	40
4.1	CI-WATER.....	40
4.1.1	Canned GSSHA	40
4.1.2	Parley’s Creek Management Tool	42
4.1.3	GSSHA Index Map Editor	44
4.1.4	Streamflow Prediction Tool.....	46
4.1.5	Observed Data.....	48
4.2	HydroShare	49
4.2.1	Snow Inspector.....	50
4.2.2	Shapefile Viewer.....	50
4.2.3	HIS Time Series Viewer	51
4.2.4	Raster Viewer.....	52
4.3	National Flood Interoperability Experiment.....	53
5	Discussion.....	55
5.1	Software Review	55
5.1.1	Spatial Database Comparison	57
5.1.2	Spatial Data Publishing Comparison	58

5.1.3	Mapping Library Comparison.....	59
5.1.4	Spatial Analysis Comparison.....	60
5.1.5	Web Development Software Comparison.....	61
5.2	Tethys Platform.....	64
5.2.1	Tethys Software Suite.....	64
5.2.2	Tethys Software Development Kit.....	65
5.2.3	Tethys Portal.....	69
6	Conclusion	70
	References.....	73
	Appendix A. Supplementary Python Libraries.....	84
A.1	MapKit.....	84
A.2	GsshaPy.....	85
	Appendix B. Software Availability.....	87
	Appendix C. Tethys Platform Documentation.....	90

LIST OF TABLES

Table 2-1. Summary of Spatial Databases Used by Earth Science Web Apps in the Literature.....	9
Table 2-2. Summary of Geospatial Data Servers Used by Earth Science Web Apps in the Literature.....	12
Table 2-3. Summary of the Mapping Libraries Used by Earth Science Web Apps in the Literature.....	14
Table 2-4. Summary of Geoprocessing Software Used by Earth Science Web Apps in the Literature.....	17
Table 2-5. Summary of Web Programming Languages Used by Earth Science Web Apps in the Literature.....	19
Table 2-6. Summary of Web Software Used by Earth Science Web Apps in the Literature.....	20
Table 3-1. Relationship Between APIs, Python Modules, and Software Components	32
Table 5-1. Comparison of the Notable Features of Spatial Databases	58
Table 5-2. Comparison of the Notable Features of Spatial Data Publishing Software.....	59
Table 5-3. Comparison of the Notable Features of Mapping Libraries.....	60
Table 5-4. Comparison of the Notable Features of Spatial Analysis Software	61
Table 5-5. Comparison of the Notable Features of Web Software.....	63

LIST OF FIGURES

Figure 1-1. Apps add value to modeling exercises by extending their use beyond the final report.....	2
Figure 3-1. The component diagram for Tethys Platform.....	26
Figure 3-2. The Tethys app project file structure for an app called "My First App".....	33
Figure 3-3. Comparison of Django URL specification with regular expressions (top) and Tethys URL specification (bottom).	34
Figure 3-4. An illustration of the layout provided by the base template for apps, which includes areas for (a) a header, (b) navigation links, (c) main content, and (d) action buttons.....	35
Figure 3-5. Example of how to configure a Map View gizmo using Python and the gizmo tag in the HTML.	36
Figure 3-6. Tethys Portal includes an app library page, which serves as the launching point for installed apps.....	39
Figure 4-1. A screenshot of the Canned GSSHA web app developed using Tethys Platform.....	42
Figure 4-2. A screenshot of the Parley's Creek Management web app developed using Tethys Platform.....	44
Figure 4-3. A screenshot of the GSSHA Index Map Editor web app developed using Tethys Platform.....	45
Figure 4-4. A screenshot of the Streamflow Prediction Tool developed using Tethys Platform.....	47
Figure 4-5. A screenshot of the Observed Data app developed using Tethys Platform.	48
Figure 4-6. A screenshot of the Snow Inspector app developed using Tethys Platform.	50
Figure 4-7. A screenshot of the Shapefile Viewer app developed using Tethys Platform.....	51
Figure 4-8. A screenshot of the HIS Time Series Viewer developed using Tethys Platform.....	52
Figure 4-9. A screenshot of the Raster Viewer developed using Tethys Platform.....	53
Figure 5-1. Comparison of programming language composition of a normal Django web project (Tethys Platform) and three Tethys apps.	67

1 INTRODUCTION

CI-WATER is a cooperative research grant funded by the National Science Foundation involving researchers from two states and four universities: Brigham Young University, the University of Utah, Utah State University, and the University of Wyoming. CI-WATER was used to establish a robust and distributed cyberinfrastructure (CI) consisting of data services, visualization tools, and a comprehensive education and outreach program that supports this integration and improves the manner in which computer models are used to support long-term planning and water resource management in the U.S. Intermountain West (Jones et al., 2014).

One of the objectives of the CI-WATER project is to enhance access to data- and computationally-intensive modeling. Hydrologic and other types of water resources model simulations are often used in decision making to analyze watershed responses to specific scenarios (Bhuyan et al., 2003; Goodrich et al., 2008; Lam et al., 2004; Miller et al., 2007; Santhi et al., 2006). However, most stakeholders and decision makers do not have the technical expertise required to properly configure a simulation for a particular scenario. The process becomes even more daunting for physics-based hydrologic models, because of the challenges of data collection and management of large spatial and temporal datasets.

Water resources web applications or “web apps” are being used to overcome many of the challenges of using hydrologic simulations in decision-making (e.g.: Demir & Krajewski, 2013; Goodrich et al., 2008; Kulkarni et al., 2014; A. Sun, 2013; Swain et al., 2015). For example, in addition to issuing a final report, a consulting firm could also create a web app for exploring new

scenarios using the model files that were developed as part of the project (see Figure 1-1). In the context of this work I define a water resources web app as a narrowly-focused, web-accessed application for performing common tasks related to hydrology and water resources modeling. In a web environment, water resources web apps can be hosted on a remote server allowing them to be accessed simultaneously by multiple users via a web interface. This also eliminates the need for the end user to procure and maintain the high performance hardware required by the models, deal with issues related to software installation and operating system incompatibilities, or monitor and install software updates. All that is needed to use a web app is an internet connection and a web browser.

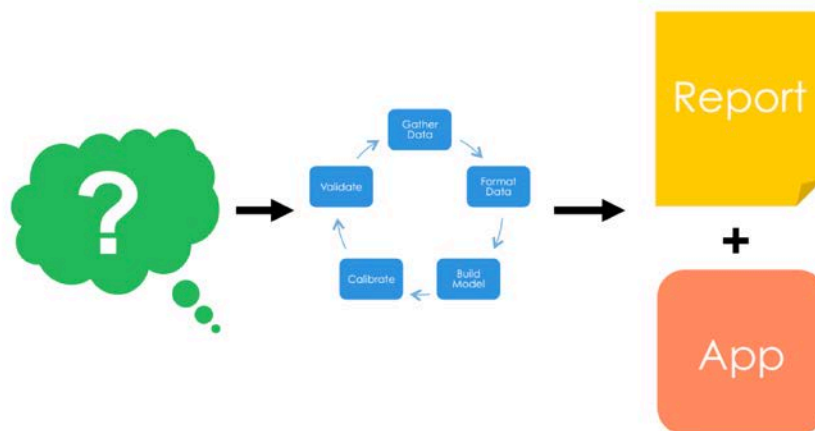


Figure 1-1. Apps add value to modeling exercises by extending their use beyond the final report.

Despite the potential the web app medium has for promoting the work of water resources scientists and engineers, the technical expertise required to develop them represents a significant barrier for would-be developers. This barrier is comprised of a series of hurdles that a novice developer would need to overcome to successfully develop a water resources web app. Many of

these hurdles require a large upfront investment of time, which deters many researchers no matter the potential benefits.

One hurdle is the need to discover and select software projects that address the spatial needs of water resources web apps. These spatial needs can be addressed using existing free and open source software (FOSS) for geographic information systems (FOSS4G), but the abundance of FOSS4G that are available can be overwhelming to new developers. For example, the Open Geospatial Consortium (OGC), an organization that creates standards for GIS software, publishes a database of over 800 registered products (Open Geospatial Consortium, 2015). Although not all of these GIS software products are for the web or even FOSS4G, understanding that distinction requires expertise that the novice developer would not have.

There are many benefits of using FOSS4G projects to address the needs of water resources web apps, but using them often requires orchestrating the use of more than one project to achieve the desired functionality. FOSS4G projects tend to be more narrowly focused in terms of functionality than their proprietary counterparts. Whereas proprietary software vendors typically offer a wide variety of GIS functionality (e.g. web mapping services, geoprocessing, and spatial storage) in a single software package, FOSS4G projects tend to focus on a single category of functionality (Steiniger & Weibel, 2010). Consequently, creating a water resources web app using FOSS4G usually requires the developer to synthesize several projects and orchestrate their use via code.

Another hurdle that developers encounter is the nature of web development. Developing a dynamic, interactive web app requires the use of multiple programming languages including HTML, CSS, JavaScript for creating the user interface, a scripting language like PHP, Python or Ruby for handling logic on the server, and frequently structured query language (SQL) for

interacting with a database. Successful web app development also requires the use of a software architectural pattern such as Model View Controller (MVC) or some variant to prevent the codebase from becoming unmanageable (Buschmann et al., 1996; Feng et al., 2011; Gamma et al., 1995; Jansson & Moon, 2001; Mason et al., 2014; Jeffrey D. Walker & Steven C. Chapra, 2014).

The objective of my dissertation research is to lower the barrier to water resources web app development with the creation of Tethys Platform, named after the Greek sea Titaness and mother of rivers. It is a development and hosting environment for water resources web apps and it effectively lowers the barrier for water resources web app development by addressing the following hurdles: (1) identify, select, and install software that meet the spatial and computational capabilities commonly required for water resources modeling; (2) orchestrate the use of multiple FOSS and FOSS4G projects and navigate their differing application programming interfaces (APIs); (3) learn the multi-language programming skills required for modern web development; and (4) develop a web-safe and fully featured web site to host the app.

Tethys Platform consists of three major components: Tethys Software Suite, Tethys Software Development Kit (SDK), and Tethys Portal. Tethys Software Suite addresses the software discovery and selection hurdle by providing a suite of FOSS and FOSS4G projects that address many of the common needs encountered in water resources web app development. It includes software for file dataset management, user account management, spatial database storage, geoprocessing, mapping and visualization, and distributed computing. Tethys SDK overcomes the web development hurdle by providing a Python MVC framework for developing water resources web apps. It also overcomes the software orchestration hurdle by providing

Application Programming Interfaces (APIs) for each element of the software suite into water resources web apps. Tethys Portal acts as the primary runtime environment for web apps developed with Tethys Platform. Built on Django, a popular Python web framework, it provides a website user interface that facilitates developing web apps and deploying completed web apps in a production setting.

A detailed description of Tethys Platform and the research backing it are presented in this work. The most significant hurdle overcome by Tethys Platform is the software selection hurdle. An extensive literature review of the FOSS4G and FOSS for web projects that back the current state-of-the-art earth sciences web apps was conducted as part of overcoming this hurdle. The results of this literature review are presented in Chapter 2. Many of the other hurdles discussed above were addressed through the development of Tethys Platform. A description of each component of Tethys Platform is included in Chapter 3. Since its inception, Tethys Platform has been used to develop a wide range of water resources web apps. Many of these web apps were created by developers working on projects other than CI-WATER including HydroShare (Tarboton et al., 2014), another NSF-sponsored project, and the National Flood Interoperability Experiment (NFIE; D. Maidment, 2014). Chapter 4 provides a description of these web apps. The conclusions of this work are presented in Chapters 5 and 6.

2 SOFTWARE REVIEW

Spatial data is an important component of water resources web apps, which makes software that facilitates spatial data use on the web a major theme of this work. These spatial needs can be addressed with geographic information systems (GIS) software. Whereas, proprietary software vendors conveniently offer all of the needed GIS functionality in a single software package, free and open source software (FOSS) projects tend to focus on a single category of functionality (Steiniger & Weibel, 2010). Thus, creating a water resources web app using FOSS requires the developer to synthesize several FOSS projects. Additionally, the abundance of FOSS for GIS (FOSS4G) projects that are available can be overwhelming to new developers. In an effort to understand the complex web of FOSS features and capabilities, I performed a review of many of the state-of-the-art FOSS software projects in the context of those that have been used by earth science web apps that have been published in the literature in the last decade (2004-2014). The scope of the review was broadened to include web apps from other disciplines in the earth sciences, because water resources web apps share many of the same specialized requirements for web development.

The FOSS projects reviewed are divided into two categories: GIS components and tools and web development software. A brief description of each software project is provided with a table of relevant earth science web apps from the literature that used the software. This chapter

aims to be an introductory guide that will save developers of water resource web apps time and effort in the genesis of their projects.

2.1 Web GIS Software Review

The vector and raster spatial data associated with hydrologic analyses require special consideration for water resources web apps. Fortunately, there is an abundance of FOSS4G available that can be used by water resources web app developers to acquire, modify, store, visualize, and analyze spatial data. However, the abundance of FOSS4G can also be overwhelming and confusing to novice developers (Steiniger & Hunter, 2012b). While many of the GIS software projects have been reviewed in the literature (Chen et al., 2010; DeVantier & Feldman, 1993; S. Li et al., 2007; Schut, 2007; Steiniger & Hunter, 2012a, 2012b; Zhao et al., 2012), my goal is to focus on the FOSS4G projects that have been selected and implemented by web app developers in the earth sciences field.

Many of the earth science web apps reviewed use software projects that implement OGC standards (OGC, 2012a). The OGC publishes specifications for data delivery over the internet such as web mapping service (OGC-WMS; OGC, 2006), web feature service (OGC-WFS; OGC, 2014) and web coverage service (OGC-WCS; OGC, 2012c) and data format standards like simple features interface standard (OGC-SFS; OGC, 2010b), geography markup language (OGC-GML; OGC, 2012b), and keyhole markup language (OGC-KML; OGC, 2008). In addition, OGC specifies standards for data search such as catalog service for the web (OGC-CSW; OGC, 2007a) and geoprocessing such as web processing service (OGC-WPS; OGC, 2007b).

The flexibility offered to developers by the interoperability of OGC compatible projects is illustrated by the fact that the system architecture of several of the reviewed projects is

described generically in terms of OGC standards rather than naming specific implementations of each standard (Han et al., 2012; Y. Li et al., 2013; X. Sun et al., 2012). For example, Han et al. (2012) describe OGC-WMS, OGC-WFS, and OGC-WCS as required components of the architecture of their system rather than specifying a specific implementation of the standards such as MapServer or GeoServer. Other projects implement custom versions of the OGC standards (Blower et al., 2013; Blower et al., 2009; Feng et al., 2011; Frehner & Brändli, 2006; Oulidi et al., 2012).

There are many types of GIS software that are tailored to specific GIS tasks (Steiniger & Hunter, 2012b; Zhao et al., 2012). For simplicity, the FOSS4G review is organized into four broad categories: spatially enabled databases for storage, spatial data publishing for sharing spatial data, mapping libraries for visualizing spatial datasets, and spatial analysis for geoprocessing and spatial algorithms.

2.1.1 Spatial Databases

Spatial databases store geographical data in a file system that is suitable for large datasets with thousands of features and provide an efficient mechanism to store, query, analyze, and update these data (Steiniger & Hunter, 2012b). Many spatial databases are extensions of existing structured query language (SQL) databases and implement the OGC-SFS standard (SQL option), which defines how spatial objects should be represented. Of the web apps reviewed, three spatially enabled SQL databases were used including: MySQL Spatial, PostGIS, and SpatiaLite. A summary of the spatial databases used by the web apps from the literature is shown in Table 2-1. A brief description of each spatial database is also provided.

Table 2-1. Summary of Spatial Databases Used by Earth Science Web Apps in the Literature

FOSS4G	Web App
SpatialLite	BASHYT (Cau et al., 2013)
MySQL Spatial	Automated Geospatial Watershed Assessment (AGWA; Goodrich et al., 2008, 2011) Flood Assessment Modeling Tool (Kulkarni et al., 2014)
PostGIS	Object-Oriented and OpenGIS Hydro Information System (3O-HIS; Leone et al., 2006) Open Source Web Fire Mapper (Davies et al., 2009) Spatial Forest Information System (S. Li et al., 2007) WebGIS for Geospatial Vector Data Sharing (Y. Fang & Feng, 2009) integrated Geospatial Urban Energy Information & Support System (iGUESS; de Sousa et al., 2012) Cloud Framework for Hydro Information System (Blagoj Delipetrev et al., 2012) Hydrogeological Information System (HydrIS; Oulidi et al., 2012) Natural Resources Information System (Singh et al., 2012) Water Management Decision Support System (EDSS; A. Sun, 2013) National Operational Assessment of Hazards (NOAH; Alconis et al., 2013) Web-based Hydrologic Transport Model (Brooking & Hunter, 2013)
PostGIS	Web Application for Water Resources (Blagoj Delipetrev et al., 2014) Iowa Flood Information System (IFIS; Demir & Krajewski, 2013) eHabitat 2.0 (Dubois et al., 2013) Emissions Inventory (Gkatzoflias et al., 2013) Web Application for Water Resources (Blagoj Delipetrev et al., 2014)

PostGIS

PostGIS is a spatial database extension for the [PostgreSQL](#) database (Holl & Plum, 2009; Nguyen, 2009a). Steiniger and Hunter (2012b) claim that PostGIS provides the most extensive implementation of the OGC-SFS standard. In addition, PostGIS boasts impressive support for raster data and analysis, for which it incorporates the GDAL library (Warmerdam, 2008) to support a wide array of raster formats. The extension provides three new column types including *geometry*, *geography*, and *raster* and it supports spatial indexing schemes for fast retrieval

(Nguyen, 2009a). PostGIS also includes a large library of spatial database functions (~ 400 in version 2.1 not including variants) for basic analysis of vector and raster objects (e.g. clip, buffer, intersection, and union), conversion between the vectors and rasters, and spatial reference system transformations.

Spatialite

Spatialite is the spatial extension for the SQLite database ([Steiniger and Hunter 2012b](#)). The project aims to be roughly equivalent to PostGIS, but far lighter weight in the SQLite fashion. It uses the geometry library of GEOS (Open Source Geospatial Foundation, 2014) to implement OGC-SFS (Zhao et al., 2012). Like PostGIS, Spatialite boasts a large library of database functions for performing spatial analysis (~ 400 in version 4.2 not counting variants). However the functions assume planar geometry and effectively ignore the spatial reference system of the data. SQLite performs well in single user environments, but it is not well equipped to handle multiple concurrent connections as occurs often in a web environment (Furieri, 2008).

MySQL Spatial

Steiniger and Hunter (2012b) state that MySQL Spatial provides a basic implementation of OGC-SFS. MySQL spatial supports vector data formats, but does not support rasters at this time. MySQL spatial also provides a database function library, though not as extensive as PostGIS or Spatialite (~ 90 functions in version 5.7). All calculations that are performed assume Euclidean (planar) geometry. The spatial types and functions can be used with several MySQL storage mechanisms including MyISAM, InnoDB, and ARCHIVE and spatial indexing is supported in MyISAM and InnoDB tables.

2.1.2 Spatial Data Publishing

Spatial data can be published using a class of software called a geospatial data server. The role of a geospatial data server is to make spatial data available in web-friendly formats. This is done by offering the data or visualizations of the data as OGC standardized web services, which can then be rendered on a web page in a browser using a mapping library or plugin (discussed in the next section). The primary OGC standards that are applicable to geospatial data servers are the web mapping service (OGC-WMS), web feature service (OGC-WFS), and web coverage service (OGC-WCS). OGC-WMS is concerned with serving raster and vector data as maps (images), whereas OGC-WFS allows direct access to the data including reading, writing, and updating. OGC-WCS is used to serve raster or imagery layers. Three FOSS4G spatial data publishing software projects were used in the earth science web apps reviewed: MapServer, GeoServer, and deegree. A summary of the geospatial data servers that are used by web apps in the literature is shown in Table 2-2.

MapServer

MapServer is a Common Gateway Interface (CGI) application written in the C programming language that can be installed on any operating system (Gkatzoflias et al., 2013; Vatsavai et al., 2006). The C implementation also gives MapServer exceptional performance compared to the Java implementations of the other projects (OSGeo, 2014). It is capable of serving spatial datasets as OGC web services including OGC-WMS, OGC-WFS, and OGC-WCS. MapServer supports numerous raster and vector data formats via the GDAL libraries including TIFF, GeoTIFF, ESRI shapefiles, and PostGIS.

Table 2-2. Summary of Geospatial Data Servers Used by Earth Science Web Apps in the Literature

FOSS4G	Web App
MapServer	Web-based Hydrologic Geographic Information System (WHYGIS; Choi, Engel, & Farnsworth, 2005) (Choi, Engel, Theller, et al., 2005)
	Open Source Web Fire Mapper (Davies et al., 2009)
	integrated Geospatial Urban Energy Information & Support System (iGUESS; de Sousa et al., 2012)
	Natural Resources Information System (Singh et al., 2012)
	BASHYT (Cau et al., 2013)
	Emissions Inventory (Gkatzoflias et al., 2013)
	Spatial Forest Information System (S. Li et al., 2007)
deegree	Spatial Forest Information System (S. Li et al., 2007)
	Hydrogeological Information System (HydRIS; Oulidi et al., 2012)
GeoServer	WebGIS for Geospatial Vector Data Sharing (Y. Fang & Feng, 2009)
	Geospatial Model Sharing Platform (GeoMSP; Feng et al., 2011)
	USGS Geo Data Portal (Blodgett et al., 2012)
	Cloud Framework for Hydro Information (Blagoj Delipetrev et al., 2012)
	Web-based Hydrologic Transport Model (Brooking & Hunter, 2013)
	Web-based groundwater database management system (Iwanaga et al., 2013)
	Web Application for Water Resources (Blagoj Delipetrev et al., 2014)
Environmental Data System (EDS; Melis et al., 2014)	

MapServer is configured via special files called Mapfiles. It also includes an Application Programming Interface (API) called MapScript that can be used to configure the server and interact with the server's data programmatically. MapScript is available for several programming languages including Python, Java, and PHP. The datasets that MapServer serves can be stored on the file system of the server or in spatially enabled databases (such as PostGIS).

GeoServer

GeoServer is a Java-based web server that implements the OGC-WFS, OGC-WCS, OGC-WMS, and OGC-WPS web service standards (Iacovella & Youngblood, 2013). As a Java application, GeoServer can be used with any of the major operating systems. It is packaged as a

web archive (WAR) for use with existing servlet container applications such as Apache Tomcat and Jetty (GeoServer, 2013).

GeoServer provides a graphical web administration tool for configuration. Alternatively, GeoServer can be configured programmatically through a Representational State Transfer (REST) interface. Other features of GeoServer include integrated OpenLayers and Google Earth™ support, GeoWebCache automated spatial caching, tile mapping, and wide support for spatial databases such as PostGIS, ArcSDE, Oracle, and DB2. GeoServer relies heavily on GeoTools (GeoTools, 2014), an open source Java library that provides GIS support for spatial data types such as vector and raster layers (Ballatore et al., 2011).

deegree

The deegree project is another Java implementation of OGC web services and can be run on all operating systems. It provides implementations of the OGC-WFS, OGC-WMS, OGC-CSW, and OGC Web Map Tile Service (OGC-WMTS; OGC, 2010a) web services. It also provides support for web processing (OGC-WPS). Like GeoServer, deegree provides a web administration tool for configuration and it offers a REST like interface for programmatically configuring the server. It supports various data sources such as PostGIS, shapefiles, and OGC-GML (Müller, 2007).

2.1.3 Mapping Libraries

Mapping libraries are needed to visualize spatial data in a web environment. Mapping libraries or plugins consume OGC-WMS, OGC-WFS, and OGC-WCS web services and render the maps for presentation in a client (i.e.: web browser). The mapping libraries used in the earth science web apps reviewed are JavaScript libraries that run in web browsers. Three mapping

libraries are included in this review: OpenLayers and Google Maps™ and Google Earth™. Table 2-3 shows a summary of the web apps from the literature that used mapping libraries.

Table 2-3. Summary of the Mapping Libraries Used by Earth Science Web Apps in the Literature

FOSS4G	Web App
OpenLayers	WebGIS for Geospatial Vector Data Sharing (Y. Fang & Feng, 2009)
	Geospatial Model Sharing Platform (GeoMSP; Feng et al., 2011)
	USGS Geo Data Portal (Blodgett et al., 2012)
	integrated Geospatial Urban Energy Information & Support System (iGUESS; de Sousa et al., 2012)
	Cloud Framework for Hydro Information (Blagoj Delipetrev et al., 2012)
	DEM Explorer (Han et al., 2012)
	Water Management Decision Support System (EDSS; A. Sun, 2013)
	Web-based Hydrologic Transport Model (Brooking & Hunter, 2013)
	Cloud Framework for Hydro Information (Blagoj Delipetrev et al., 2012)
	Web-based groundwater database management system (Iwanaga et al., 2013)
Flood Assessment Modeling Tool (Kulkarni et al., 2014)	
Google Earth and/or Maps	Web GIS Based Hydrograph Analysis Tool (WHAT; Lim et al., 2005)
	Geoportal for Hydrological Applications (Díaz et al., 2008)
	Automated Geospatial Watershed Assessment (AGWA; Goodrich et al., 2008, 2011)
	GODIVA2 (Blower et al., 2009)
	Forest Fires Online/Offline Mapping and Monitoring Application (FOMA; Carvalheiro et al., 2010)
	Available WATER Resource (AWARE; Granell et al., 2010)
	Virtual Sensor System (Hill et al., 2011)
	Novel Google Earth Visualizing (X. Sun et al., 2012)
	National Operational Assessment of Hazards (NOAH; Alconis et al., 2013)
	Fire Logic Animation (FLogA; Bogdos & Manolakos, 2013)
	Iowa Flood Information System (IFIS; Demir & Krajewski, 2013)
	Emissions Inventory (Gkatzoflias et al., 2013)
	Environmental Data System (EDS; Melis et al., 2014)
CyberFlood (Wan et al., 2014)	
Combination	ncWMS (Blower et al., 2013)
	Web Application for Water Resources (Blagoj Delipetrev et al., 2014)

OpenLayers

OpenLayers is a web-mapping client library for rendering interactive maps on a web page (Hazzard, 2011). It is a pure JavaScript library for building rich web-based geospatial applications similar to Google Maps™. OpenLayers is capable of rendering vector and raster data from a variety of formats including GeoJSON, OGC-KML, OGC-GML, and OGC web services. It leverages WebGL and Canvas 2D for better performance. OpenLayers also provides methods for drawing on the map and editing data interactively. It allows developers to use a variety of services for base maps including Open Street Map, Bing, MapQuest, and Google. OpenLayers does not currently support a 3D globe type environment. It does not require a plugin and does not have the use restrictions that are imposed by the Google license (Steiniger & Hunter, 2012b), although using some of the proprietary base maps (e.g.: Google and Bing) in OpenLayers may invoke licensing restrictions.

Google Earth™

While Google Earth™ is not FOSS it is considered in this review because of its popularity and with some restrictions it is free of cost for most users. It is not free for commercial use and private users are limited to 25,000 map requests per day (Steiniger & Hunter, 2012b). One notable feature of Google Earth™ is the ability to easily animate and display data in a 3D globe environment using the OGC-KML format. The disadvantage to using Google Earth™ as a map renderer is that it requires the user to install a browser plugin, which is not supported on all operating systems or 64-bit web browsers. Additionally, the Google Earth API for the web plugin has been deprecated as of December 12, 2014 and will lose support completely December 14, 2015 (Google, 2014a).

Google Maps™

Google Maps™ provides a 2D mapping environment with high-resolution base map imagery. The new version of Google Maps™ (version 3) provides a library that allows users to draw shapes on the map and edit spatial data interactively. Like Google Earth™, Google Maps™ is capable of displaying spatial data in OGC-KML format. Alternatively, data can be added dynamically using the JavaScript API. Unlike Google Earth™, Google Maps™ does not require a browser plugin.

2.1.4 Spatial Analysis

The FOSS software projects that were used to support spatial analysis in the earth science web apps reviewed are presented in this section. Spatial analysis in water resources web apps can be achieved by using software projects that implement the OGC-WPS standard. An OGC-WPS can be installed on a stand-alone server that is optimized for geoprocessing, which tend to be designed to handle multiple simultaneous requests and a heavy processing load. Note that the deegree and GeoServer projects that were discussed in the spatial data publishing also include OGC-WPS functionality. Although it is unclear whether the web apps that used GeoServer and deegree used the OGC-WPS features, both projects will be described in this section. A summary of the web apps that used geoprocessing web service software is shown in Table 2-4.

52° North WPS

The 52° North WPS project represents a full implementation of the OGC-WPS standard (52°North, 2014; Schut, 2007). 52°North WPS provides an extensible, pluggable framework for publishing geoprocessing algorithms as web services. It can be linked with existing geoprocessing libraries such as GRASS (GRASS Development Team, 2014), Sextante (Olaya &

Gimenez, 2011), and ArcGIS® Server for out-of-the-box geoprocessing capabilities (Steiniger & Hunter, 2012b). 52°North WPS also allows developers to publish custom Python scripts (Sanner, 1999), R scripts (John Chambers, 2013), and Java processes as web services. A number of geospatial data types are supported as input such as GeoTiff, ArcGrid, Shapefiles, OGC-GML, and OGC-KML, and OGC data services (OGC-WMS, OGC-WFS, and OGC-WCS). All results can be stored as simple web accessible resources or as OGC web services.

Table 2-4. Summary of Geoprocessing Software Used by Earth Science Web Apps in the Literature

FOSS4G	Web App
52°North WPS	Geoportal for Hydrological Applications (Díaz et al., 2008)
	Available WATER Resource (AWARE; Granell et al., 2010)
	USGS Geo Data Portal (Blodgett et al., 2012)
PyWPS	integrated Geospatial Urban Energy Information & Support System (iGUESS; de Sousa et al., 2012)
	eHabitat 2.0 (Dubois et al., 2013)
	Modeling Web Services via OGC-WPS (Castronova et al., 2013)

PyWPS

PyWPS is an implementation of OGC-WPS written in Python. Like any OGC-WPS, PyWPS does not process data itself, rather, it provides the link between the web and the local tools on the server such as GRASS, GDAL, and R scripts. Castronova et al. (2013) implemented an instance of PyWPS to demonstrate how the OGC-WPS standard can be extended to offer scientific modeling as a web service.

GeoServer WPS

GeoServer provides a full implementation of OGC-WPS in addition to the spatial data publishing services. The processes can be called with GeoServer resources as inputs and they can

output to new GeoServer resources. GeoServer provides the JTS Topology Suite (Vivid Solutions, 2014) for default geoprocessing capabilities and it allows for custom processes written in Java.

deegree WPS

The deegree project also offers an implementation of OGC-WPS. No default processes are provided with deegree WPS as of version 3.2.0. However, it does provide a mechanism for publishing custom Java processes. In older documentation, deegree promised connections to GRASS, Sextante, and a proprietary processing library called FME (Safe Software, 2014).

2.2 Web Development Software Review

Water resources web apps require a strategy for developing the web interface and synthesizing all of the software components. As a minimum, they require a web server and HTML for building the web pages of the web app. However, a scripting language on the server is often required to handle interaction with database, other software, and other logic of the web app. The web development software used by the earth science web apps reviewed is summarized in this section. The web development software covered in this section is organized into hierarchical layers of programming language, web frameworks, and content management systems. A summary of web programming languages and software used to implement the web apps in the literature is shown in Table 2-5 and Table 2-6.

Table 2-5. Summary of Web Programming Languages Used by Earth Science Web Apps in the Literature

Category	FOSS	Web App
Programming Languages	PHP	SICI Hydrological and Geomorphological Catastrophe Information System (Guzzetti & Tonelli, 2004)
		Cloud Framework for Hydro Information System (Blagoj Delipetrev et al., 2012)
		Natural Resources Information System (Singh et al., 2012)
		Novel Google Earth Visualizing (X. Sun et al., 2012)
		Fire Logic Animation (FLogA; Bogdos & Manolakos, 2013)
		Iowa Flood Information System (IFIS; Demir & Krajewski, 2013)
		Emissions Inventory (Gkatzoflias et al., 2013)
	Web Application for Water Resources (Blagoj Delipetrev et al., 2014)	
	Java	Virtual Database for Distributed Ecological Data (Frehner & Brändli, 2006)
		Object-Oriented and OpenGIS Hydro Information System (3O-HIS; Leone et al., 2006)
		USDA Conservation Reserve Program DSS (Rao et al., 2007)
		Geoportal for Hydrological Applications (Díaz et al., 2008)
		Open Source Web Fire Mapper (Davies et al., 2009)
		WebGIS for Geospatial Vector Data Sharing (Y. Fang & Feng, 2009)
		Web-based Participatory Wind Energy Planning (WePWEP; Simao et al., 2009)
		Available WATER Resource (AWARE; Granell et al., 2010)
		SPARROW DSS (Booth et al., 2011)
		Geospatial Model Sharing Platform (GeoMSP; Feng et al., 2011)
		DEM Explorer (Han et al., 2012)
Hydrogeological Information System (HydrIS; Oulidi et al., 2012)		
Custom OGC-WMS Implementation for NetCDF files (Blower et al., 2013)		
Web-based Hydrologic Transport Model (Brooking & Hunter, 2013)		
Snowmelt Flood Early Warning System (S. Fang et al., 2013)		
Flood Assessment Modeling Tool (Kulkarni et al., 2014)		
PERL	Web-based Hydrologic Geographic Information System (WHYGIS; Choi, Engel, & Farnsworth, 2005)	

Table 2-6. Summary of Web Software Used by Earth Science Web Apps in the Literature

Category	FOSS	Web App
Web Frameworks	Ruby on Rails	integrated Geospatial Urban Energy Information & Support System (iGUESS; de Sousa et al., 2012)
	CodeIgniter (PHP)	Automated Geospatial Watershed Assessment (AGWA; Goodrich et al., 2008, 2011)
	Django (Python)	Water Management Decision Support System (EDSS; A. Sun, 2013) <u>ubertool</u> (Flaishans et al., 2014)
	Backbone.js (Client Side)	Web-based Interactive River Model (WIRM; Jeffrey D Walker & Steven C Chapra, 2014)
Content Management Systems	CKAN	Water quality InformaTion System (WAITS; Peres et al., 2013)
	Drupal	Web-based groundwater database management system (Iwanaga et al., 2013)
		Web-based water infrastructure database (WATERiD; Jung et al., 2013) Earth Science Environmental Simulator (ESES; Van Knowe et al., 2014)

2.2.1 Programming Languages

Programming languages can be used in web development to make websites more dynamic and to handle advanced logic beyond simply returning static HTML. For example, programming languages could be used to execute simulation runs or prepare a complex visualization when a user submits a request. Practically any programming language could be used to fulfill this purpose, provided the language provides mechanisms for working with HTTP requests. The following section provides a summary of the programming languages that were used in earth science web app development.

PHP

PHP (recursive acronym for PHP: Hypertext Preprocessor) is a very popular scripting language that is especially suited for web development. PHP is embedded in the HTML pages of the website and the code is executed on the server when the web page is requested. It includes support for a wide range of databases via ODBC and provides a database abstraction layer called PDO. It also has many extensions that add common web functionality such as managing sessions and cookies, user authentication, and file uploads (Royappa, 2000).

Java

A significant number of projects used Java-based solutions for web development. Several developers used Java Platform, Enterprise Edition (Java EE, formerly J2EE) and JavaServer Pages (JSP) to develop their earth science web apps, while others did not specify the Java framework used. These Java technologies are free and open source and provide a powerful, cross-platform development environment for creating and running large-scale, multi-tiered, scalable, reliable, and secure network applications (Oracle, 2012).

2.2.2 Web Frameworks

Web frameworks provide a scriptable approach for building websites with the intent of alleviating much of the low-level coding typically associated with static website design. A web framework typically offers features for accessing databases, building pages from dynamic templates, managing users and sessions, and creating a secure website. Most web frameworks follow some form of the Model View Controller (MVC) development paradigm, where the model consists of the data of the website (often a database model), the view is the presentation of

the data, and the controller provides the logic that interprets the data for the view and handles user input.

Python Web Frameworks

The scientific modules such as SciPy and NumPy has made Python a popular scripting language for scientific computing (Millman & Aivazis, 2011; Oliphant, 2007). As such, Python-powered web frameworks are a popular choice for building earth science web apps. There are over fifty Python web frameworks, but the most popular are Django, Grok, Pylons, TurboGears, web2py, and Zope2 (The Python Wiki, 2014). Python web frameworks vary greatly in default functionality. For example, Django and TurboGears provide a significant amount of functionality that is enabled by default to make development quicker and easier, while Pylons provides only minimal default functionality to allow greater flexibility for the developer.

CodeIgniter

CodeIgniter is an MVC PHP web framework with exceptional performance and virtually no configuration. Some of the features of CodeIgniter include full-featured database classes, form and data validation, security and cross-site filtering, session management, and file uploading. It is an attractive option for web developers who already use PHP, but want to use a structured framework approach (Upton, 2007).

Ruby on Rails

Ruby on Rails, or simply Rails, is a web framework written in the Ruby language. Rails web applications are organized using the MVC pattern like the other web frameworks. Rails features convention over configuration, meaning that it will do a lot of the heavy lifting of web development automatically. Many other web frameworks are influenced by Rails development.

There are tens of thousands of sites developed using Rails including Twitter and GitHub (Tate & Hibbs, 2006).

Client Side Frameworks

To avoid the delay caused by frequent interaction between the browser/client and the server, some web apps are developed to run completely in the client (web browser) as pure JavaScript applications. The initial request downloads the source code for the application from the server and initiates the web app. Jeffrey D Walker and Steven C Chapra (2014) developed a client-side web app that runs the Web-based Interactive River Model (WIRM). The web app was developed using Backbone.js (Sugrue, 2013), a JavaScript MVC client-side framework. They used the Python web framework Django on the server to handle user authentication and database interaction.

2.2.3 Content Management Systems

Content Management Systems (CMS) are often built on a web framework and provide a higher level of abstraction to web development. CMS web sites use a GUI in the browser with a limited amount of coding. The focus of a CMS is to allow the developers to manage content independently according to the web template that is chosen. The user interface of a CMS has a front-end and back-end structure where the front-end is accessible to users and only the administrators can access the back-end for maintaining and development purposes (Rojas-Sola et al., 2011). This type of system is ideal for non-technical administrators of the website.

Drupal

Drupal is a widely used FOSS CMS. It provides a browser-based graphical user interface to develop a website minimizing the need to write code. It comes with only basic functionality

enabled. Developers add functionality to a website by installing modules from an extensive library. Custom modules can be created using PHP and a series of hooks into other routines provided by the Drupal developers. Once the site development is complete, end users can easily maintain and update the site using the same interface (Drupal, 2013).

CKAN

CKAN is a specialized CMS for hosting datasets using a built-in data management system (CKAN, 2013). CKAN is built on the Pylons Python web framework. The data management system comes ready to host data out of the box and the data can be stored with a rich set of metadata. CKAN also provides a set of Python programming interfaces for building custom extensions and a REST API for uploading and downloading data programmatically (CKAN, 2013).

3 TETHYS PLATFORM

Tethys Platform consists of three primary components: Tethys Software Suite, Tethys SDK, and Tethys Portal. Before describing each of these components independently, it is helpful to see how they all fit into the bigger picture of Tethys Platform and to understand the distinction I draw between “platform” and “framework”.

A platform provides a complete runtime environment for software, whereas a framework provides code and patterns to facilitate the development of software. In many cases platforms provide frameworks to promote use of the platform. Though many users do not realize it, an operating system is a good example of a platform. The operating system provides an environment in which applications like a web browser or PDF reader can be run. The developers of these applications use the frameworks and APIs provided by the operating system (e.g.: .NET framework for Windows or Cocoa API for Mac) to build the applications that will run in the operating system runtime environment. Similarly, Tethys Portal and Tethys Software Suite make up a runtime environment for water resources web apps and Tethys SDK provides the framework that developers use to create web apps with APIs that allow developers to tap into the functionality of Tethys Software Suite. Figure 3-1 displays the major components of Tethys Platform.

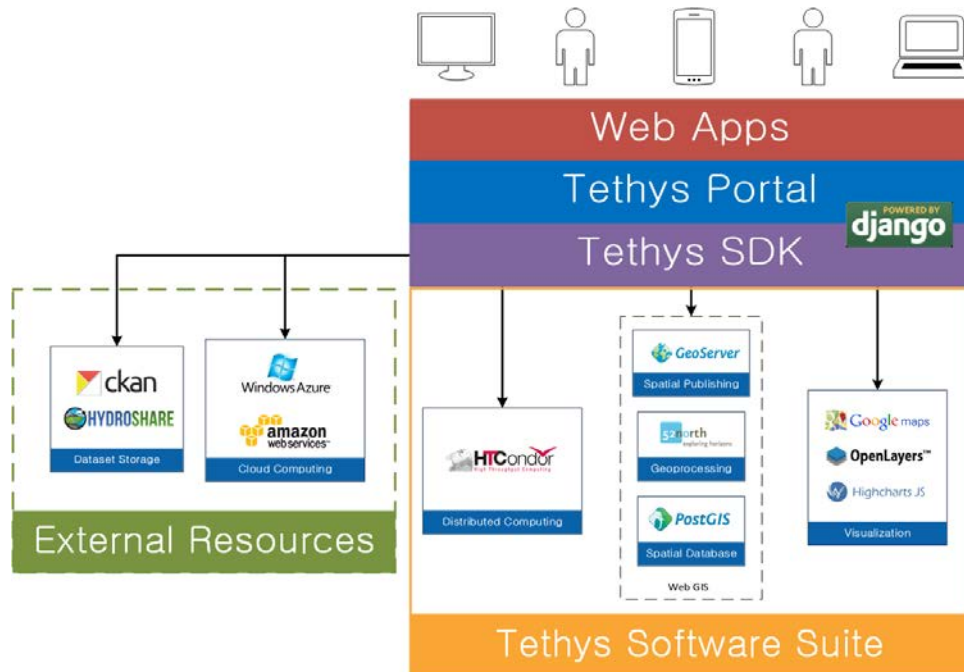


Figure 3-1. The component diagram for Tethys Platform.

3.1 Tethys Software Suite

Tethys Software Suite is the component of Tethys Platform that provides access to resources and functionality that are commonly required to develop water resources web apps. The primary motivation of creating the Tethys Software Suite was to address the software discovery and selection hurdle discussed previously. Some of the more specialized needs water resources app must provide arise from the spatial data components of the models that are used in the apps. Distributed hydrologic models, for example, are parameterized using raster or vector layers such as land use maps, digital elevation models, and rainfall intensity grids. Consequently, a majority of the software projects included in the software suite are web GIS projects that can be used to acquire, modify, store, visualize, and analyze spatial data. Tethys Software Suite includes other software projects to address computing and visualization needs of water resources

web apps. Sections 3.1.1 to 3.1.6 describe the included software in terms of the functionality they provide to water resources web app developers. Section 3.1.7 describes the strategy that is taken to reduce the burden of installing the many requisite software projects.

3.1.1 Spatial Database

The PostgreSQL database with PostGIS, a spatial database extension (Holl & Plum, 2009; Nguyen, 2009b) provides spatial data storage capabilities for Tethys web apps. PostGIS adds spatial field types including raster, geometry, and geography. PostGIS provides an extensive implementation of the applicable OGC standards (Steiniger & Hunter, 2012b). The extension also provides database functions for basic analysis of GIS objects and coordinate transformation. It also supports spatial indexing schemes that allow for quick retrieval of records from large spatial tables during query.

3.1.2 Geoprocessing

52°North WPS is included as one means for supporting geoprocessing needs in water resources web app development. 52°North WPS is a full open-source implementation of the OGC-WPS standard (52°North, 2014; Schut, 2007). It provides an extensible, pluggable framework for publishing geoprocessing algorithms as web services and it can be linked with geoprocessing libraries such as GRASS (GRASS Development Team, 2014), Sextante (Olaya & Gimenez, 2011), and ArcGIS® Server (ESRI, 2004) for out-of-the-box geoprocessing capabilities (Steiniger & Hunter, 2012b). 52°North WPS also allows developers to publish custom Python (Python Software Foundation, 2013) and R (J. Chambers, 2013) scripts as web services.

The included PostGIS extension can also provide geoprocessing capabilities on data that are stored in spatially-enabled databases. PostGIS includes SQL geoprocessing functions for splicing, dicing, morphing, reclassifying, and collecting/unioning raster and vector types. It also includes functions for vectorizing rasters, clipping rasters with vectors, and running stats on rasters by geometric region (Holl & Plum, 2009).

3.1.3 Map Rendering

There are two capabilities needed to visualize spatial data in a web application: a map server, and a mapping library or plugin for the browser. The role of a map server is to render the data in web friendly formats (e.g.: PNG, KML, GML, and GeoJSON) and publish the data as standardized web services. Mapping libraries and plugins access the data that are published on map servers via the web service API and render the data as interactive maps.

GeoServer is included for publishing spatial data as web services. GeoServer is a Java-based program, implemented with OGC web service standards including Web Map Service (OGC-WMS), Web Feature Services (OGC-WFS), and Web Coverage Service (OGC-WCS; Iacovella & Youngblood, 2013). This allows the data to be retrieved in a variety of web-friendly formats regardless of the format of the data stored on the server. GeoServer also implements the OGC-Styled Layer Descriptor (OGC-SLD) standard to provide a mechanism for styling layers. It is capable of serving many common spatial files types including Shapefiles, ArcGRID, GeoTIFF and others and it can be used to publish spatial database tables from PostGIS.

3.1.4 Visualization

Two alternatives for displaying visualizations of spatial datasets in apps are included: OpenLayers and Google Maps™. OpenLayers is a JavaScript web-mapping client library

(Steiniger & Hunter, 2012b) for rendering interactive maps on a web page. (Hazzard, 2011). It is capable of displaying 2D maps of OGC web services and a myriad of other spatial formats. Google Maps™ provides the ability to render spatial data in a 2D mapping environment similar to OpenLayers (Google, 2014b), but it only supports displaying data in KML formats and data that are added via JavaScript API. Both maps provide a mechanism for drawing on the map for user input.

Plotting capabilities are provided by Highcharts, a JavaScript library created by Highsoft AS. The plots created using Highcharts are interactive with hovering effects, pan and zoom capabilities, and the ability to export the plots as images. Supported plots include line, spline, area, area spline, column, bar, pie, scatter, angular gauges, area range, area spline range, column range, bubble, box plot, error bars, funnel, waterfall and polar chart types (Highsoft AS, 2014).

3.1.5 Distributed Computing

To facilitate the large-scale computing that is often required by water resources applications, Tethys Software Suite leverages the computing management middleware HTCondor. HTCondor is both a resource management and a job scheduling software. It was developed at the University of Wisconsin-Madison with the primary goal of scavenging idle computing time on networked desktop workstations (Litzkow et al., 1988). It has matured to be a flexible and powerful computing resource management system that can make use of supercomputers, computing grids, and cloud computing. HTCondor facilitates High Throughput Computing (HTC). HTC differs from High Performance Computing (HPC) in that its main objective is to provide a large amount of computing power over a long period of time (days to months) whereas HPC focuses on providing a large amount of computing power per second (Livny et al., 1997). HTC systems are very well suited to performing loosely coupled or

uncoupled (embarrassingly parallel) tasks. These types of computing tasks, which include stochastic analysis and parameter sweeps, are commonly encountered in water resources modeling.

3.1.6 File Dataset Storage

CKAN and HydroShare are used to address flat file storage needs through the Tethys SDK which is discussed in more detail later. CKAN is an open source data sharing platform that streamlines publishing, sharing, finding, and using data. There is no central CKAN hub or portal, rather data publishers setup their own instance of CKAN to host the data for their organization. CKAN has become a popular mechanism for governments around the world to share data. Some examples include data.gov, data.gov.uk, PublicData.eu, Helsinki Region Infoshare, and IATI Registry (Open Knowledge Foundation, 2014).

HydroShare is an online hydrologic model and data sharing portal being developed by the Consortium of Universities for the Advancement Hydrologic Science, Incorporated (CUAHSI). It builds on the sharing capabilities of CUAHSI's Hydrologic Information System by adding support for sharing models and using social media functionality to enhance collaboration over datasets and models (Tarboton et al., 2014).

3.1.7 Docker Installation

Tethys Software Suite includes many software projects with esoteric installation instructions, which could make installation of Tethys Platform a burden. To overcome this potential hurdle, Docker images have been developed for PostgreSQL with the PostGIS extension, GeoServer, and 52°North WPS. Docker builds on Linux virtualization capabilities to provide a lightweight mechanism for packaging and distributing web applications (Docker Inc.,

2015). Docker images are used to create containers, which are essentially stripped down virtual machines running only the software included in the image. Unlike virtual machines, the Docker containers do not partition the resources of your computer (processors, RAM, storage), but instead run as processes with full access to the resources of the computer.

Three Docker images are provided as part of Tethys Software Suite including an image for PostgreSQL with PostGIS, 52° North WPS, and GeoServer. The installation procedure for each software has been encapsulated in a Docker image reducing the installation procedure to three simple steps: (1) install Docker, (2) download the Docker images, (3) and create Docker containers (i.e. deploy the Docker images).

3.2 Tethys Software Development Kit

Tethys SDK is the Tethys Platform component that includes a framework for developing web apps, a command line interface (CLI) for simplifying common management tasks, and APIs for interfacing with each component of Tethys Software Suite. Tethys SDK is needed to address the web development and software orchestration hurdles discussed previously. It is built on top of Django, a high-level Python web framework that can be used to rapidly develop websites (Django Software Foundation, 2015). The major components of Tethys SDK are discussed in the following sections. The relationship between the various APIs, Python modules, and the elements of the software suite is shown in Table 3-1.

Table 3-1. Relationship Between APIs, Python Modules, and Software Components

API	Python Modules	Software Components
Template Gizmos	Django templating language	OpenLayers Google Maps™ HighCharts
Dataset Services	<i>tethys_dataset_services</i>	CKAN HydroShare
Spatial Dataset Services	<i>tethys_dataset_services</i> gsconfig	GeoServer
Persistent Stores	SQLAlchemy	PostgreSQL
Spatial Persistent Stores	SQLAlchemy GeoAlchemy	PostgreSQL w/ PostGIS
Web Processing Services	OWSLib	52° North WPS
Distributed Computing	<i>CondorPy</i> <i>TethysCluster</i>	HTCondor Amazon Web Services Microsoft Azure

Note: The names of custom Python modules developed for Tethys Platform are italicized.

3.2.1 App Development Framework

The Tethys app development framework builds on the Django web framework to provide a structured approach for building and configuring apps. They are developed using the model-view-controller (MVC) software architectural pattern, where the model is composed of the data management components of the app (e.g.: data model), the view is composed of the user interface and web pages of the app, and the controller houses the logic of the app. A typical Tethys web app project is organized with a specific file structure (Figure 3-2). The Tethys CLI includes a scaffolding command that is used to automatically generate new app projects with the required files and structure.

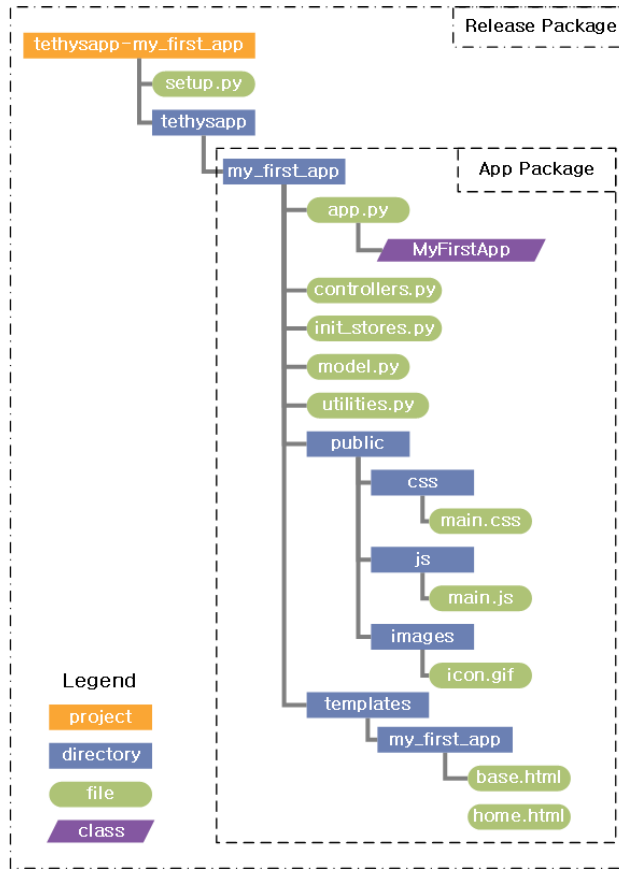


Figure 3-2. The Tethys app project file structure for an app called "My First App".

Tethys app project structure consists of two primary elements: the release package and the app package. The release package contains the files needed to distribute and install the Tethys app including a setup script and the app package. The app package contains the source code for the app. The app configuration file (`app.py`) includes a class that is used to configure the app project. The other files included in the app package reflect the MVC architectural pattern that is employed in Tethys app development (e.g.: `models.py`, `templates`, and `controllers.py`).

Much of Tethys app development uses the structures and conventions provided by Django, but Tethys includes a layer designed to simplify some aspects of Django development. For example, the mechanism for designing URLs with variables in Django requires the use of

regular expressions—a sequence of characters that form a search pattern using a syntax that would prove difficult for inexperienced developers. Tethys provides a layer of abstraction to simplify URL design using a simple syntax. Figure 3-3 illustrates this difference in approach to URL design.

```
1 # Example of Django URL
2 r'^/users/(?P<user_id>[0-9]+)/edit/$'
3
4 # Equivilent URL using Tethys SDK
5 '/users/{user_id}/edit'
```

Figure 3-3. Comparison of Django URL specification with regular expressions (top) and Tethys URL specification (bottom).

The views, or web pages use the Django templating language. Rather than requiring developers to start from scratch with each template, the Tethys framework provides a base template that includes a standard layout for app pages with areas for a header, navigation links, action buttons, and primary content (see Figure 3-4). This reduces the amount of repetitive template coding required for developing a web app in Tethys.

3.2.2 Command Line Interface

The Tethys SDK provides a command line interface to automate some of the common management tasks associated with developing web apps with Tethys Platform. Tethys CLI includes a scaffolding command for generating new app projects, a command for managing app databases, commands to assist with installation and updating Tethys platform, and commands for managing the software suite Docker images and containers.

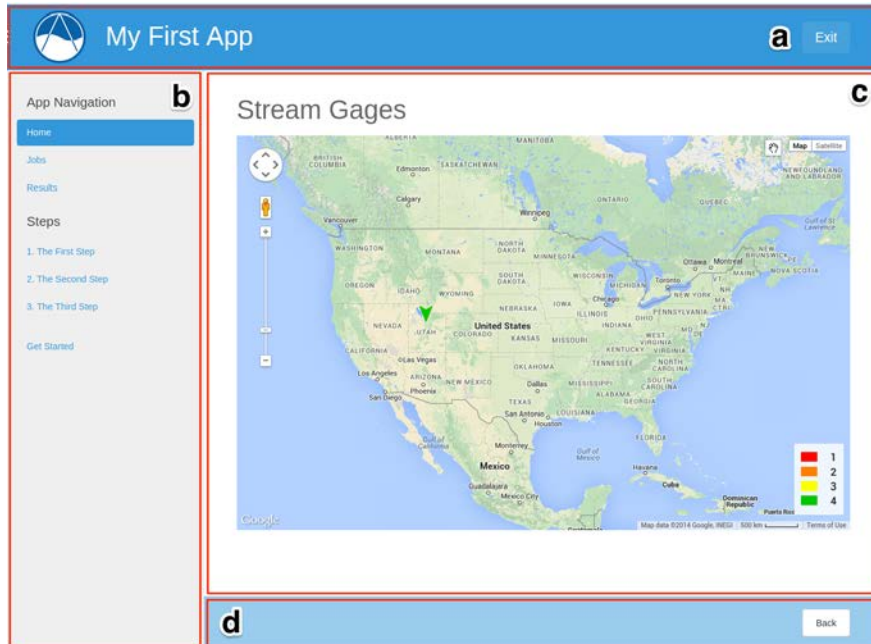


Figure 3-4. An illustration of the layout provided by the base template for apps, which includes areas for (a) a header, (b) navigation links, (c) main content, and (d) action buttons.

3.2.3 Template Gizmos API

The Template Gizmos API simplifies the development of the user interface of an app by providing a set of common elements or “gizmos” that can be re-used to make Tethys apps interactive. Using this API, developers can add date-pickers, plots, and maps to their apps with minimal coding. Gizmos are configured using Python in the controller of a page and inserted using a custom Django template tag. This reduces the need to work in multiple languages by automatically inserting the necessary HTML, CSS, and JavaScript in place of the gizmo template tag. Figure 3-5 shows an example of how to create a map using the Map View gizmo.

```

1 # Configure in Controller
2 map_view_options = {'height': '400px',
3                     'width': '100%',
4                     'controls': ['ZoomSlider',
5                                 'Rotate',
6                                 'FullScreen',
7                                 'ScaleLine',
8                                 {'ZoomToExtent': {'projection': 'EPSG:4326',
9                                                  'extent': [-135, 22, -55, 54]}},
10                                {'MousePosition': {'projection': 'EPSG:4326'}}},
11                     ],
12                     'layers': [{'WMS': {'url': 'http://demo.opengeo.org/geoserver/wms',
13                                         'params': {'LAYERS': 'topp:states'},
14                                         'serverType': 'geoserver'}
15                                }],
16                     ],
17                     'view': {'projection': 'EPSG:4326',
18                              'center': [-100, 40], 'zoom': 3.5,
19                              'maxZoom': 18, 'minZoom': 3},
20                     'base_map': 'OpenStreetMap'
21 }
22
23 # Add this line to HTML Template
24 {% gizmo map_view map_view_options %}

```

Figure 3-5. Example of how to configure a Map View gizmo using Python and the gizmo tag in the HTML.

3.2.4 Persistent Stores APIs

The SDK includes two APIs for interacting with the PostgreSQL database software component of Tethys Platform: the Persistent Store API and the Spatial Persistent Store API. These APIs streamline the use of SQL databases (termed persistent stores) in Tethys apps. Using these APIs developers can easily create one or more PostgreSQL databases for each app. The Persistent Store API includes SQLAlchemy, a Python module that provides object-relational mapping capabilities that allow developers to create data models for their databases using an object-oriented programming approach (Bayer, 2015). The Spatial Persistent Store API adds spatial capabilities to the Persistent Store API by adding the PostGIS extension to the databases and providing GeoAlchemy2, a Python module that adds spatial support to SQLAlchemy (Lemoine, 2015). The Persistent Store APIs include a command in Tethys CLI that is used to manage databases for apps. This command automates common database management tasks such

as creating the databases and tables, updating databases with changes, and initializing the databases with data.

3.2.5 Dataset Services APIs

The Dataset Services and Spatial Dataset Services APIs provide a means of managing flat files in apps. The Dataset Services API can be used to store or access file datasets in a CKAN instance or on HydroShare. It provides custom Python wrappers for the REST APIs of both services allowing developers to interact with the services using Python methods, rather than constructing REST calls.

The Spatial Dataset Services API provides a Python module for working with the GeoServer software component. Using the Spatial Dataset Services API developers can programmatically add spatial file datasets such as Shapefiles and GeoTIFF files to GeoServer and retrieve them to construct maps. The data served by GeoServer can be accessed in a variety of web-friendly formats using the OGC-WFS and OGC-WMS web services.

The Python wrappers for CKAN, HydroShare, and GeoServer are all implemented in a module called `tethys_dataset_services`, which is available on the Python Package Index. This allows these interfaces to be used in scripts and applications outside of Tethys Platform.

3.2.6 Web Processing Services API

The Web Processing Services API provides a Python interface for the 52° North Web Processing Service software component of Tethys Platform via a third-party module called OWSLib (Kralidis, 2010). Developers can use this API to execute geoprocessing methods that are made available as Web Processing Services. WPS is an OGC standard that specifies how inputs and outputs for processing services should be handled.

3.2.7 Distributed Computing API

The distributed computing API consists of two Python libraries: CondorPy and TethysCluster. CondorPy interfaces with HTCondor and provides a mechanism for defining and submitting high throughput computing jobs. To support distributed computing, an HTCondor pool of computing resources must be configured. This can be done using on-premise resources, or TethysCluster can be used to provision computing resources using the Amazon Web Services or Microsoft Azure commercial clouds. TethysCluster automates the process of provisioning a cluster of virtual machines and configuring them with HTCondor so that they can serve as a distributed computing system.

3.3 Tethys Portal

Tethys Portal is the Tethys Platform component that acts as the primary runtime environment for Tethys web apps. It overcomes the hurdles associated with starting new websites and deploying websites. It is implemented as a Django website project and it leverages the capabilities of Django to provide the core website functionality that is often taken for granted in modern web applications. It includes a user account system complete with user profiles and a password reset mechanism for forgotten passwords. It also provides a landing page that can be used to showcase the capabilities of the Tethys Platform instance and an app library page that serves as the access point for installed apps (Figure 3-6). It includes an administrator backend that can be used to manage user accounts, permissions, link to elements of the software suite, and customize the instance.

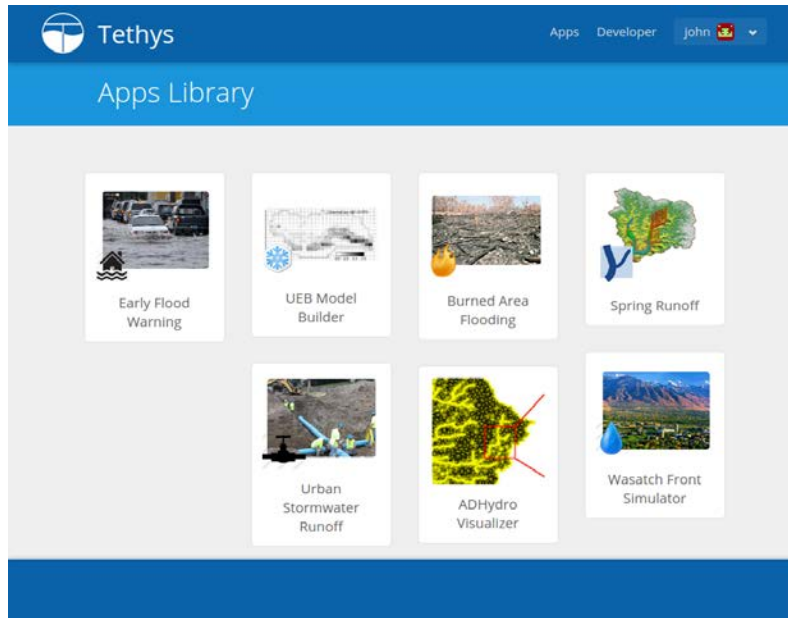


Figure 3-6. Tethys Portal includes an app library page, which serves as the launching point for installed apps.

As the runtime environment, any organization that wishes to deploy water resources web apps that are developed using Tethys Platform will need to stand up their own instance of Tethys Portal. With this in mind, Tethys Portal was designed to be easily customizable and rebranded to fit the needs of the organization. The title, theme, logos, and content on the homepage can easily be changed through the administrative backend.

4 APPLICATIONS

Tethys Platform was designed to lower the barrier to development for scientists and engineers who wish to convey their data and models via interactive web apps. Nine web apps that were developed using Tethys Platform are presented in this section to demonstrate its utility. A brief description of each web app is provided and the Tethys Platform capabilities that each app implements are discussed.

4.1 CI-WATER

This section describes web apps that were developed as part of the objectives of CI-WATER. Of interest to CI-WATER was the development of several prototype web apps that could be used to demonstrate how the web app medium can be used to enable greater access to cyber infrastructure and modeling resources. Developed concurrently with Tethys Platform, most of these web apps were also used as part of the process of discovery and development of its features.

4.1.1 Canned GSSHA

The Canned GSSHA web app was developed to demonstrate the Canned Modeling method of flood forecasting (Dolder et al., 2015). The premise of the Canned Modeling method is that when a flood event is imminent, there is limited time to execute hydrologic model runs in

an attempt to predict the outcome of the flood event. The solution provided by the Canned Modeling method is to pre-run a large number of models with varying input parameters, called "scenarios", and store or "can" the results for lookup in the time of a crisis. When a potential flood event occurs, the canned model database is queried using the current or forecasted conditions and the model run with the closest match is instantly returned. The Canned Modeling method can be applied to any hydrologic model, or even any combination of different hydrologic models.

A Gridded Surface Subsurface Hydrologic Analysis (GSSHA; Downer & Ogden, 2004) model was developed for a 2.5 km² test watershed, and seven input parameters were selected to generate scenarios that would produce both snowmelt and rain-driven floods. A Latin Hypercube approach was used to create different combinations of the selected input parameters uniformly over the entire parameter space resulting in 2187 scenarios. The GSSHA model was executed for each of the generated scenarios and the resulting hydrographs were stored in a database.

The Canned GSSHA app provides an intuitive, single-page user interface that allows users to alter the observed or forecasted input parameters and view the computed hydrograph of the closest match (Figure 4-1). A cluster of sliders on the left-hand side of the screen can be used to modify the input parameters. The polar plot at the center shows normalized values of the parameters selected by the user in yellow and the normalized values of the parameters from the pre-computed model that most closely matches the user input in green. The plot on the right-hand side of the screen displays the hydrograph of the closest matching model. Each time the user changes the value of any of the sliders, the lookup is near instantaneous and the polar plot and hydrograph update immediately.



Figure 4-1. A screenshot of the Canned GSSHA web app developed using Tethys Platform.

The Canned GSSHA app makes use of Template Gizmos API to create the Range Sliders and the Plot Views. A minimal amount of JavaScript was required to handle the dynamic plot updates when the user changes the values of the sliders. The app also uses the Persistent Store API to create a PostgreSQL database that stores the scenarios parameters and hydrographs of the model runs. The Canned GSSHA app has proven valuable not only as an illustration of the Canned Modeling approach but also as an educational tool to help students understand hydrology.

4.1.2 Parley's Creek Management Tool

The Parley's Creek Management Tool was developed to make water management research products more accessible to decision makers (Erfan Goharian & Burian, 2014). The Parley's Watershed is one of four major drainages that are included in the Salt Lake City, Utah, USA protected watershed canyons (USDA Forest Service, 2015). The Parley's Creek Basin, located on western slope of Wasatch Mountains, includes two reservoirs, Little Dell and

Mountain Dell. The reservoirs were developed with the primary use of municipal and industrial water supply and secondary use of flood control. The main inflows to the reservoirs are generated from Lambs Creek and Dell Creek. Moreover, the outflow from Little Dell reservoir discharges into Mountain Dell. The bypassed water from Parley's water treatment facility flows into the Parley's Creek and passes through the urbanized area of Salt Lake City into the Jordan River and ends at the Great Salt Lake.

E. Goharian, Burian, Bardsley, et al. (2015) worked with Parley's Creek reservoir managers to develop a system dynamics model using GoldSim, a Monte-Carlo simulation software for dynamic modeling of complex systems (GoldSim Technology Group, 2015). The model allows managers and stakeholders to explore various management scenarios for the Parley's Creek system. It also allows managers to explore the impact of various climate change projections on reliability and vulnerability of system (E. Goharian, Burian, & Karamouz, 2015).

The Parley's Creek Management Tool web app provides a simple user interface and structured workflow for the Parleys Creek Management GoldSim model. The workflow consists of 4 or 5 pages (depending on the type of climate scenario selected) that prompt the user to select a climate scenario, modify reservoir characteristics, adjust the inflow as a multiplier of the historical average on a monthly basis, adjust demand rates on a monthly basis, and view a summary of the parameters. The GoldSim model is hosted as a web service using 52° North WPS, which can be executed from a page in the web app. After a model run has been completed the user can view several plots and download the results as an excel spreadsheet from a results page (shown in Figure 4-2). The results include the inflow to the reservoirs, volume of water in reservoirs, releases, and spills as well as the reliability of system to meet the water demand from Salt Lake City.

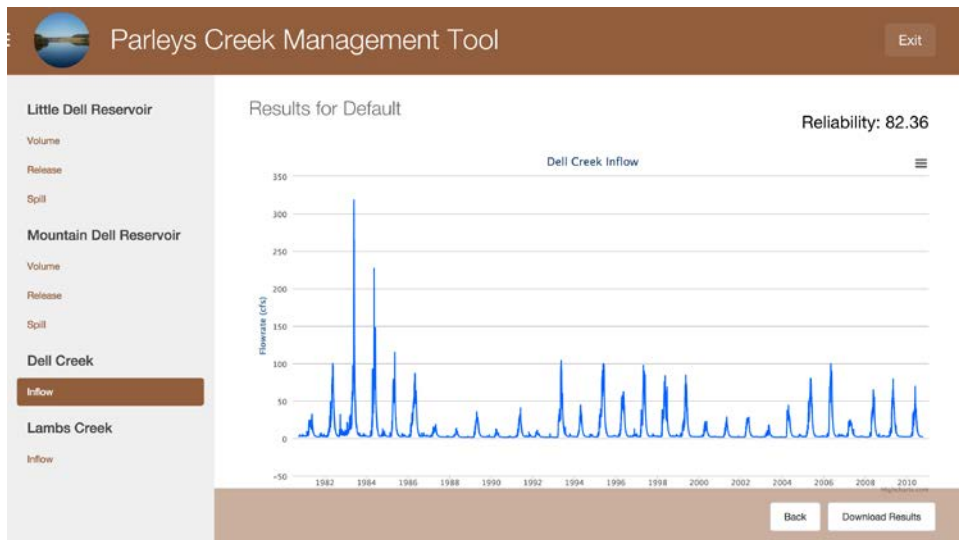


Figure 4-2. A screenshot of the Parley's Creek Management web app developed using Tethys Platform.

The Parley's Creek Management Tool was developed using the Template Gizmos API for UI elements including the Text Input, Range Slider, and Plot View gizmos. The app also makes use of the Persistent Store API to provision databases for storing jobs and results. The execution of the GoldSim model web service is handled using OWSLib, a third-party Python module that is included as part of the Web Processing Services API.

4.1.3 GSSHA Index Map Editor

The GSSHA model represents spatially varying watershed characteristics such as land use or soil properties are represented in GSSHA using GRASS ASCII rasters also called index maps. A common analysis that is performed using GSSHA is a land use change analysis from a pre-calibrated model that may be prompted by a proposal for developing an undeveloped parcel.

Anderson (2014) used Tethys Platform to develop a web app called “GSSHA Index Map Editor” that streamlines land use change analysis (see Figure 4-3). Upon launching the app, the

user is prompted to select a previously uploaded GSSHA model package or upload their own. The user selects the index map they wish to edit, which is displayed using an editable Google Maps™ map. The user can edit the index map in one of three ways including manually drawing the areas affected by the change, uploading a shapefile, or merging two existing index maps. After editing the map the user is guided through a series of steps for assigning hydrologic characteristics for the changed areas. Finally, the user can save the scenario, run it, and view the results.

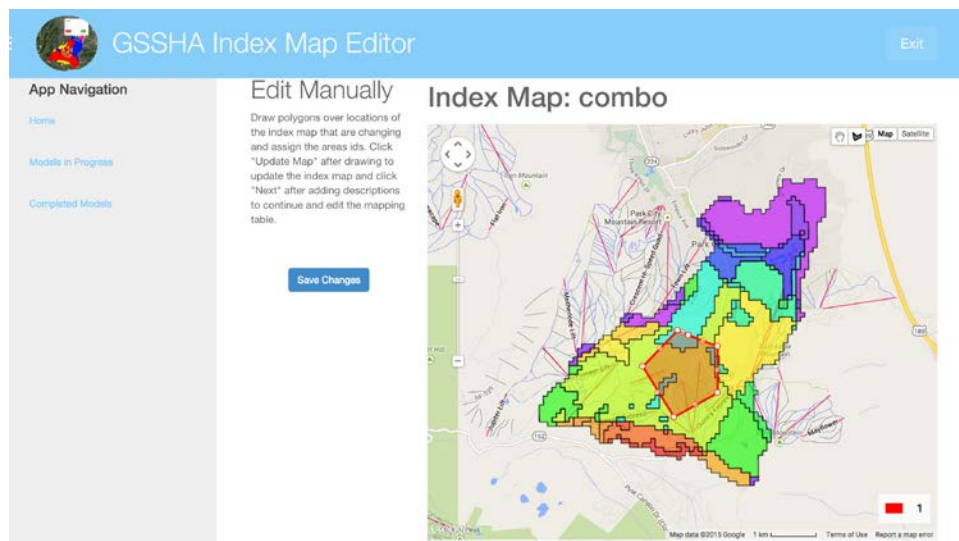


Figure 4-3. A screenshot of the GSSHA Index Map Editor web app developed using Tethys Platform.

The GSSHA Index Map Editor web app uses the Spatial Persistent Store API to provision PostgreSQL databases with the PostGIS extension activated. The geoprocessing tasks are performed using PostGIS database functions. When the user selects a GSSHA model package, it is read into the PostGIS database using GsshaPy, a custom Python module that interfaces between the GSSHA model files and spatially-enabled databases. Modifications to the model are

recorded in the database and the model is written back to file to allow the updated model to be executed.

The app demonstrates the use of several gizmos from the Template Gizmos API including the Google Map View and Map View gizmos. The app uses the Dataset Services API to manage the GSSHA model packages, which are stored as simple zip archives of model files on a CKAN server. The GSSHA models are executed using a web service which is called using the Web Processing Services API.

4.1.4 Streamflow Prediction Tool

Snow (2015) developed a web service that automatically queries the latest global runoff forecasts published by the European Center for Medium-Range Weather Forecasts (ECMWF), downscales the forecast to a higher resolution catchments using Esri's RAPID Toolbox, and routes the resulting runoff through a high resolution stream network using the Routing Application for Parallel computation of Discharge (RAPID) model (David et al., 2011). The result is a high-resolution dataset of two-week stream flow forecasts every twelve hours with the potential for nationwide or even global coverage.

Snow also developed a web app using Tethys Platform to view the recent streamflow forecasts called the Streamflow Prediction Tool. The computations are performed via the web service twice daily as the ECMWF runoff prediction datasets become available. After the computations are finished, they are then deposited into a data store such as HydroShare or CKAN. The web app automatically retrieves the most recent week's worth of predictions from the data store for visualization by the user.

Upon launching the app, users are prompted to select one or more watersheds to display on a map, from which they can select a reach and view a time series of the forecasted flow two

weeks in advance. The ECMWF forecast is an ensemble of 52 different scenarios, so results are displayed as a statistical hydrograph (e.g.: min, max, mean) of the ensemble forecast for any selected reach in the stream network (Figure 4-4).

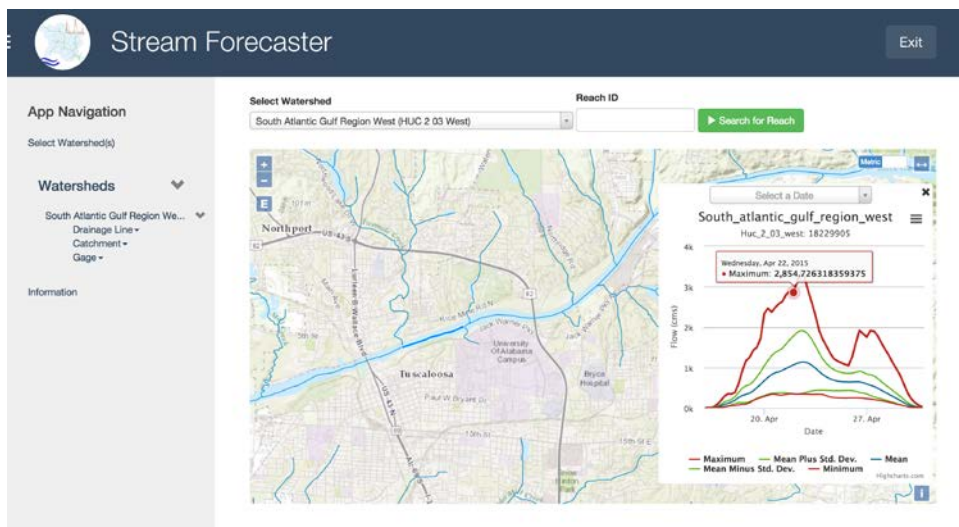


Figure 4-4. A screenshot of the Streamflow Prediction Tool developed using Tethys Platform.

The Streamflow Prediction Tool web app uses the GeoServer component of the software suite via the Spatial Dataset Services API to serve the stream network, catchment, and USGS gauges maps for each watershed. The app is able to offload dataset management to an instance of CKAN using the Dataset Services API. The Template Gizmos API is used to provide many of the user interface elements including the Button Group, Toggle Switch, Select Input, and Text Input gizmos. While the app implements OpenLayers to provide the mapping, it was developed prior to the creation of the Map View Gizmo. Streamflow Prediction Tool makes use of the Persistent Store API to provide a database for settings and to manage information relevant to the watersheds.

4.1.5 Observed Data

Saguibo (2015) developed a web app that is capable of consuming data feeds from HydroServers called Observed Data Figure 4-5. HydroServer is a computer server that allows hydrologic data producers to publish data as web services and maintain local control of the data (Horsburgh et al., 2010). It is one component of the Hydrologic Information System (HIS) that has been developed by CUAHSI to enable a web services approach to hydrologic data sharing (D. Maidment et al., 2004). Data that obtained via HydroServers is transferred over the Internet using an extensible markup language (XML) standard called WaterML (Valentine et al., 2012). The Observed Data web app was designed to act as a client for HydroServers. It discovers available HydroServers through the HIS Central catalog (<http://hiscentral.cuahsi.org/>), queries the servers for the desired hydrologic parameters, and displays visualizations of the time series datasets on interactive web plots.

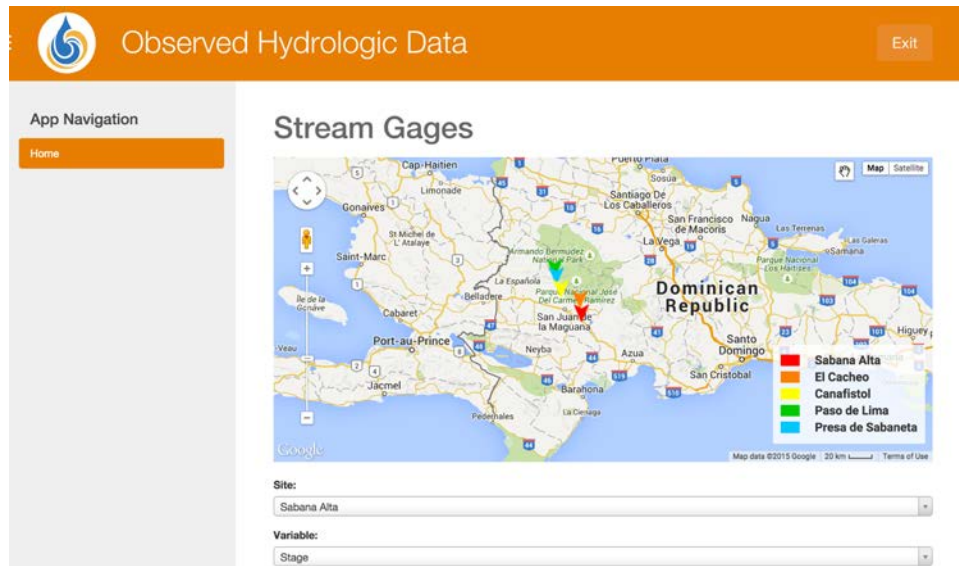


Figure 4-5. A screenshot of the Observed Data app developed using Tethys Platform.

The Observed Data app consists of two pages that leverage the Template Gizmos API. The first page includes a Google Map View gizmo that displays the locations of available HydroServers and a form for defining the query to be performed. The second page displays the time series plot of the parameter selected using a Plot View gizmo.

This app is unique among the apps developed by the CI-WATER team, because it is the first app to be developed after the first stable release of Tethys Platform allowing it to leverage features that were either rapidly evolving or non-existent for the other apps. The app is also notable, because it was developed by a developer who matched the profile of the anticipated developers of Tethys apps: an individual with engineering background, limited scientific scripting and GIS experience, and minimum knowledge of web development. Saguibo estimates that he invested 25 hours at the outset to install Tethys Platform review the tutorials and other documentation with the actual development of the Observed Data app requiring an additional 100 hours.

4.2 HydroShare

HydroShare is an online hydrologic model and data-sharing portal that leverages social media functionality to enhance collaboration over datasets and models (Tarboton et al., 2014). One objective of HydroShare is to provide a set of web-based tools that can be used to visualize and analyze the datasets they host. The HydroShare team has adopted Tethys Platform as one mechanism for creating and hosting many of these web-based tools. They have deployed an instance of Tethys Platform at <http://apps.hydroshare.org/> that will host their apps. This section will describe some of the Tethys apps that have been developed by the HydroShare team.

4.2.1 Snow Inspector

The Snow Inspector app was developed to enable the easy discovery and access of the snow cover history that is derived from the MODIS Terra daily snow cover dataset for any point on Earth (Hall et al., 2006). The user can use the interactive map to pick a location, retrieve the snow data, display the time series, and download the processed data in WaterML or CSV file format (Figure 4-6). The data source used by this app is the NASA's Global Image Browse Services (GIBS; Cechini et al., 2013; Thompson et al., 2014) Web Map Tile Service (WMTS) server. The original dataset is the MOD10A1 fractional snow cover grid with 500 meter resolution. The app depends on the open-source pypng library for extracting values from the WMTS images.

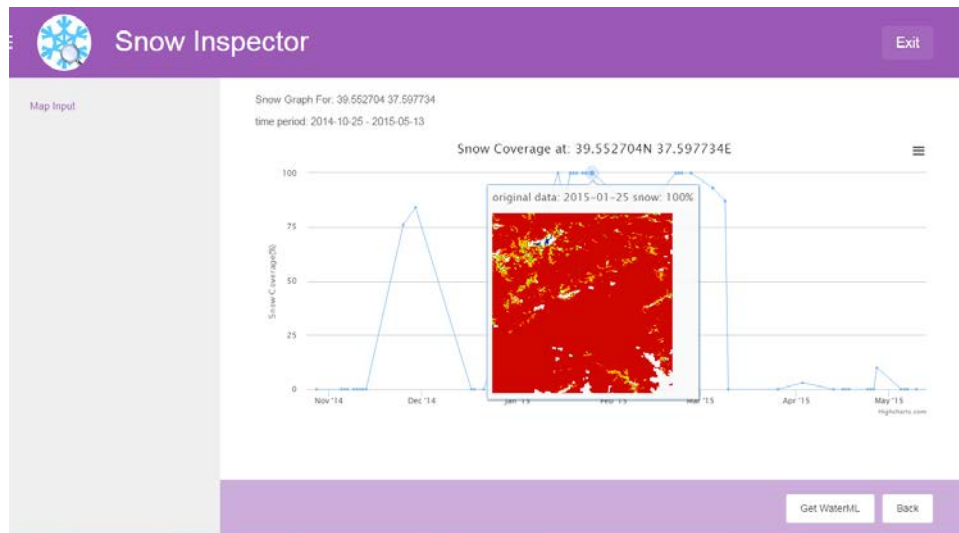


Figure 4-6. A screenshot of the Snow Inspector app developed using Tethys Platform.

4.2.2 Shapefile Viewer

The Shapefile Viewer app has been created as a proof of concept app that demonstrates how Shapefiles stored in HydroShare can be pushed into a Tethys app and displayed. The app

several metadata fields associated with HydroShare datasets including the resource id, file name and branch name to retrieve the dataset. Eventually, datasets in HydroShare will include a button that can be used to launch the Tethys app and provide visualization for the dataset. The shapefile is displayed using an OpenLayers map via the Map View Gizmo (Figure 4-7).

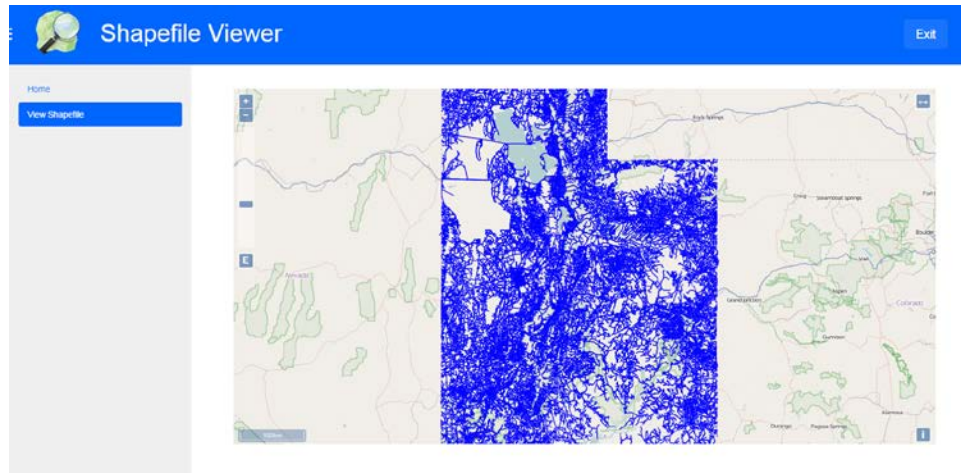


Figure 4-7. A screenshot of the Shapefile Viewer app developed using Tethys Platform.

4.2.3 HIS Time Series Viewer

This app was designed for visualizing the HIS Referenced Time Series (REFTS) resource type (Figure 4-8). REFTS resources are used to store time series datasets in HydroShare. They support data that is stored in CSV, WaterML 1.0, and WaterML 2.0 formats. The current app supports OGC WaterML 2.0 standard to enable it to process the contents of REFTS datasets. Upon launching the app, the user supplies a HydroShare resource id and filename of an REFTS as a URL query string and chooses whether to retrieve the dataset using either a HTTP GET or POST request. The app depends on the 3rd-party python library lxml which is installed automatically when the app is installed.

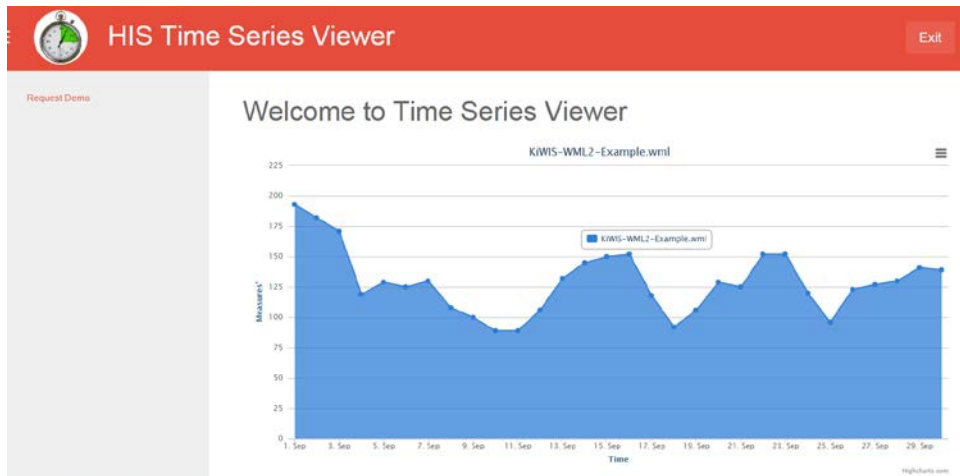


Figure 4-8. A screenshot of the HIS Time Series Viewer developed using Tethys Platform.

4.2.4 Raster Viewer

This app was designed for visualizing the HydroShare Geographic Raster resource type. A Geographic Raster resource typically contains a GeoTiff file. The viewer requires users to supply a resource id and a geotiff filename. The app retrieves the GeoTIFF file from HydroShare and transfers it to the GeoServer included in Tethys Platform using the Spatial Dataset Services API. The GeoTIFF is published as a GeoTIFF raster layer and displayed using an OpenLayers map via a Map View gizmo (Figure 4-9).

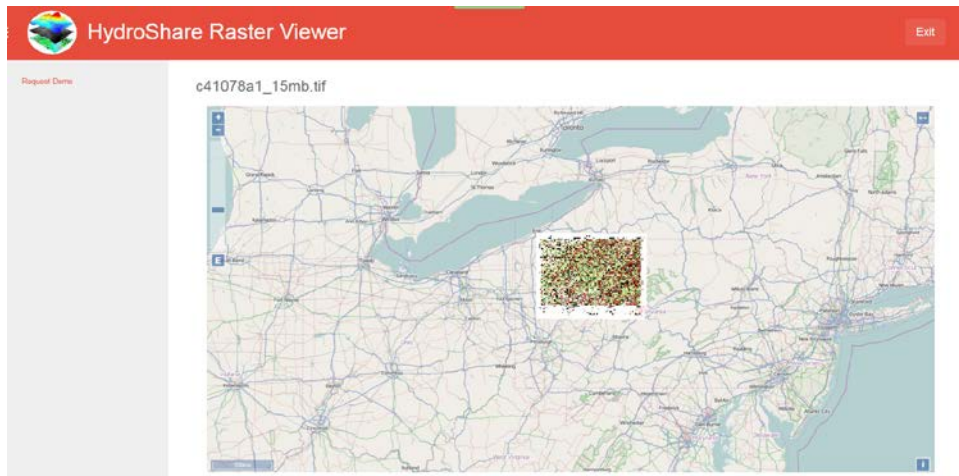


Figure 4-9. A screenshot of the Raster Viewer developed using Tethys Platform.

4.3 National Flood Interoperability Experiment

The National Flood Interoperability Experiment (NFIE) is a one-year research collaboration among government, academia, and industry partners seeking to demonstrate the next generation of national flood hydrology and emergency response. Graduate students will participate in a seven-week Summer Institute in June and July of 2015 at the National Water Center in Tuscaloosa, Alabama, USA. Participants in the Summer Institute will leverage new high spatial resolution hydrologic forecasting systems to carry out hydraulic modeling and flood inundation mapping to improve flood emergency response at the local level. If successful, this experiment will result in an increase in spatial density of flood forecasting locations by more than 700 times compared with the present NWS river forecasting system. This would be transformative for improved real-time flood information in the United States (D. R. Maidment, 2014, 2015).

Tethys Platform has been selected as one of the premier technologies that will be promoted as a tool for the 50 participants to use during the NFIE. A two-day workshop on

Tethys Platform was taught during the initial weeks of the Summer Institute. I anticipate some of the existing Tethys apps will be expanded (e.g.: Streamflow Prediction Tool) and it is likely that several new apps will be developed to achieve NFIE objectives.

5 DISCUSSION

The research conducted for this dissertation resulted in two primary products that effectively lower the barrier to water resources web app development: (1) the literature review of free and open sources software (i.e. software review) and (2) Tethys Platform. The software review addresses the software selection hurdle by significantly reducing the long list of candidate FOSS web and FOSS4G projects that could be used to provide the web and GIS capabilities for these types of apps. It also aggregates descriptions of projects that have been used to develop web apps that have been published in the peer-reviewed literature. The findings of the software review are discussed at length in Section 5.1. Tethys Platform builds on the results of the software review to further lower the barrier by providing a software suite composed of elements discovered through the software review and providing Python APIs to orchestrate the use of each software element. It also addresses many other hurdles, which will be discussed in Section 5.2.

5.1 Software Review

The free and open source software projects included in the software review were selected based on a literature review of 45 earth science web apps that were developed in the last decade (2004-2014). The FOSS projects presented in this review do not represent a comprehensive or even representative sampling of all FOSS web GIS and web development software available. Rather, the long list of available FOSS software projects was narrowed to only those projects that

have been used by existing earth science web apps that have been published in recent peer-reviewed literature. This review extends previous FOSS4G and FOSS web software reviews by focusing on only those FOSS projects that have been tried and proven in existing earth science web apps.

The quality and capabilities of the earth science web apps included in the review vary significantly. Some of the earth science web apps were developed as prototype or demonstration systems (e.g.: Bogdos & Manolakos, 2013; Feng et al., 2011; Oulidi et al., 2012), while others were developed as full-featured data and modeling services that were currently in operation at the time of writing (e.g.: Alconis et al., 2013; Blodgett et al., 2012; Demir & Krajewski, 2013). The earth science web apps address data and modeling needs in a wide range of applications including water resources, wild fires, water quality, urban planning, flood warning, ecology, and geology. Of all earth science web apps used in the review, approximately 80% were published in the last 5 years (2009-2014) and almost 45% were published in the last 2 years indicating a growing interest in web apps as a medium for earth science modeling and data.

Each earth science web app in the review included at least one FOSS software component, with a majority of the earth science web app projects using several FOSS projects to address various spatial data needs. Web apps that included proprietary components in addition to at least one FOSS component were included. In several cases, web apps were included in which the GIS capabilities were provided entirely by proprietary software or were not specified, but the web software was a FOSS solution (S. Fang et al., 2013; Flaishans et al., 2014; Frehner & Brändli, 2006; Rao et al., 2007; Simao et al., 2009; Van Knowe et al., 2014; Jeffrey D Walker & Steven C Chapra, 2014). The remaining discussion will focus on each category of FOSS reviewed.

5.1.1 Spatial Database Comparison

No other category of FOSS software exhibited as strong a preference for one project as the spatial database category. Of the earth science web apps reviewed, 19 reported using an SQL database with a spatial extension. The PostgreSQL database with the PostGIS spatial extension was overwhelmingly the preferred solution with 15 web apps using PostGIS. MySQL with Spatial extension was used by 3 web apps and 1 web app used SQLite with the SpatiaLite extension.

It was not unexpected that SQLite with SpatiaLite wasn't as popular with the earth science web apps reviewed, because SQLite is suboptimal for web environments as discussed in Section 2.1.1. It is surprising that MySQL Spatial was not selected as often, because MySQL is the most popular FOSS SQL database for general web development. However, the spatial implementation of PostGIS is superior on several fronts. One difference is that PostGIS has extensive support for raster data, while MySQL Spatial has no raster support. Another key difference is that PostGIS boasts a library of about 400 database functions (not counting variants) to perform spatial analysis on both raster and geometry columns, whereas the spatial function library of MySQL Spatial is minimal with only about 90 functions. The other primary difference is that MySQL spatial functions ignore the spatial reference system and use only Euclidean (planar) distances while PostGIS has support for spatial reference systems. A summary of the notable features of the spatial databases reviewed is provided in Table 5-1.

Table 5-1. Comparison of the Notable Features of Spatial Databases

Spatial Database	Number Web Apps	Spatial Functions	OGC-SFS	Vector Format	Raster Format	Spatial Reference Calculations	Concurrent Access
PostGIS	15	~ 400	x	x	x	x	x
MySQL Spatial	3	~ 90	x	x			x
Spatialite	1	~ 400	x	x			

5.1.2 Spatial Data Publishing Comparison

The line was not so clearly drawn in the other categories. Of all the earth science web apps reviewed, 16 reported using software for spatial data publishing. MapServer was used in 7 web apps, GeoServer was used in 8 web apps, and deegree was used in 2 web apps (one web app used both MapServer and deegree).

The three software projects are comparable in terms of their implementations of applicable OGC standards. GeoServer and deegree provide web interfaces for configuring the data on the server making them more user-friendly than the file-based configuration of MapServer. However, MapServer can be configured programmatically via MapScript in a number of different development environments including, PHP, Python, Perl, Ruby, Java, and .NET. GeoServer and deegree can be configured programmatically via REST APIs. In terms of performance, MapServer tends to outperform GeoServer and deegree by virtue of its C implementation. GeoServer and deegree also provide a WPS implantation that can be used for geoprocessing capabilities. A summary of the notable features of the spatial data publishing software included in the review is shown in Table 5-2.

Table 5-2. Comparison of the Notable Features of Spatial Data Publishing Software

Spatial Data Publishing	Number Web Apps	Implementation Language	OGC-WFS	OGC-WMS	OGC-WCS	OGC-WPS	Web Configuration	File Configuration	REST API	Scripting API
MapServer	7	C	x	x	x			x		x
GeoServer	8	Java	x	x	x	x	x		x	
deegree	2	Java	x	x		x	x		x	

5.1.3 Mapping Library Comparison

There were 24 earth science web apps that reported using mapping libraries with 12 web apps that used Google Maps™ or Google Earth™, 10 web apps that used OpenLayers, and 2 web apps that used a combination of OpenLayers and the Google libraries. The Google mapping libraries are frequently used because many people are familiar with the popular Google mapping service (<https://www.google.com/maps>). The primary advantage of Google Earth™ over Google Maps™ or OpenLayers, is that it provides a 3D-globe mapping environment. In terms of supported data formats, Google Maps™ and Google Earth™ are limited—only accepting data in OGC-KML format or via the JavaScript APIs. OpenLayers boasts support for a wide range of formats including OGC-KML, OGC-GML, GeoJSON, OGC mapping services such as OGC-WMS and OGC-WFS, and many others. Finally, OpenLayers and Google Maps™ allow for interactive user input via drawing on the map. Table 5-3 shows a comparison of the notable features of the mapping libraries presented in the review.

Table 5-3. Comparison of the Notable Features of Mapping Libraries

Mapping Library	Number Web Apps	Input Formats							Plugin Required	Draw on Map
		OGC-KML	OGC-GML	OGC-WFS	OGC-WMS	GeoJSON	3D Globe	2D Map		
Google Earth™	14	x						x	x	
Google Maps™	-	x						x	x	
OpenLayers	12	x	x	x	x	x		x	x	

5.1.4 Spatial Analysis Comparison

The spatial analysis software was the least used category of software with only six earth science web apps that specified using spatial analysis software. The 52° North WPS project was used by three of the web apps and PyWPS was used by three web apps. It should be noted that many web apps cited using GeoServer and deegree for spatial publishing, but it was unclear whether any of those projects made use of the GeoServer and deegree OGC-WPS functionality. All four software projects implement the OGC-WPS specification.

The primary difference between the implementations is in the default processes that are supported. The 52° North WPS offers the most processes “out-of-the-box” with the ability to link to the GRASS, Sextante and ArcGIS Server geoprocessing libraries as well as custom processes written in Python, R, and Java. PyWPS only supports processes written in Python and R, but it can be linked to GRASS via the Python GRASS bridge. GeoServer WPS offers the JTS Topology Suite processes in the default configuration and allows developers to write custom processes written in Java. The current version of deegree WPS only supports custom processes written in Java, though connections to GRASS, Sextante, and FME are in the works. As spatial

data publishing projects, GeoServer and deegree have the advantage of being able to operate on data that is stored locally and store the results as OGC web services, resulting in fewer file transfers. The 52° North WPS is capable of storing results as OGC web services, though it is unclear if it is able to do so without the aid of GeoServer or deegree. A summary of the notable features of the spatial analysis software reviewed is shown in Table 5-4.

Table 5-4. Comparison of the Notable Features of Spatial Analysis Software

Spatial Analysis	Number Web Apps	Implementation Language	OGC-WPS	GRASS GIS	Sextante	ArcGIS Server	GDAL	PROJ	JTS Topology	Python Scripts	R Scripts	Java Processes	Local Data
52° North WPS	3	Java	x	x	x	x				x	x	x	
PyWPS	3	Python	x	x			x	x		x	x		
GeoServer	-	Java	x						x			x	x
deegree	-	Java	x									x	x

5.1.5 Web Development Software Comparison

The web software category was the most widely varying category with at least 9 different FOSS software projects or languages that were used by earth science web app developers. Although all of the applications were web apps and necessitated some web development strategy, only 34 specified what web software was employed. The most popular web development strategy used to create the earth science web apps reviewed were Java frameworks—numbering 17 in all. The next most popular approach was to use PHP web development with eight earth science web apps that used this method. Of the remaining earth science web apps, one web app used Perl as a scripting language on the server, two web apps used the Django Python web framework, one

web app used the Ruby on Rails web framework, one web app used the CodeIgniter PHP web framework, two web apps used the Drupal content management system, one web app used the CKAN data management system, and one web app used a client-side framework called Backbone.js in conjunction with Django. It is important to note that the web development software presented in this review is not a comprehensive sampling of all the web development software available.

Most web frameworks provide strategies for solving common web development challenges such as user management, database interaction, creating dynamic HTML, and handling file uploads. The primary difference between web software lies in the approach that the web framework takes to solving web development tasks. For example, Drupal and CKAN provide user management systems that require virtually no configuration—complete with login, logout, forgotten password, and user profile pages. Django also provides a user management system, but the burden of creating the login, logout, and user profile web pages and logic rests on the developer. Neither approach is better than the other.

Consequently, selecting a web framework depends largely on the needs and complexity of the project and the preferences of the developer. Some of the factors to consider when selecting a web framework include the programming experience of the developer, the supporting libraries available (e.g.: geoprocessing libraries), the size or scale of the project, and the functionality required by the project.

In terms of the current review, Java frameworks were likely the most popular for earth science web apps for a few reasons. Many of the popular software implementations of OGC standards are Java implementations (e.g.: GeoServer, deegree, and 52° North WPS). There are mature GIS libraries available for Java including GeoTools and JTS Topology Suite. It is not

surprising that many of the web apps used PHP for web development, as PHP is part of the LAMP (Linux, Apache, MySQL, PHP) stack of software that is used to power many websites. After Java and PHP, Python was the next most popular approach. Python has recently gained traction as a scripting language in the earth sciences, making Python web frameworks a natural choice for earth science web apps. Developers with limited programming experience may consider using a system that uses a graphical approach to web design such as Drupal, while more experienced programmers may wish to use a scripting language like PHP or a Python web framework. Table 5-5 show a summary of some of the notable features of the FOSS web software reviewed.

Table 5-5. Comparison of the Notable Features of Web Software

Web Software	Number Web Apps	Language	MVC or Similar	Server Language	CMS	GUI Configuration
Java Framework	17	Java	x			
PHP	8	PHP		x		
Perl	1	Perl		x		
Django	3	Python	x			
Ruby on Rails	1	Ruby	x			
CodeIgniter	1	PHP	x			
Drupal	2	PHP			x	x
CKAN	1	Python	x		x	x
Backbone.js	1	JavaScript	x			

5.2 Tethys Platform

Tethys Platform overcomes many of the hurdles that deter scientists and engineers from using the web app medium. I have assumed the primary developers that would use Tethys Platform would be ambitious scientist and engineers with some scientific scripting experience and knowledge of but little experience with web development. In this section a detailed discussion is provided about specific technical hurdles that are overcome for developers using Tethys Platform to create water resources web apps. The discussion will be organized by the three main components of Tethys Platform: Tethys Software Suite, Tethys SDK, and Tethys Portal.

5.2.1 Tethys Software Suite

The two hurdles overcome by the Tethys Software Suite include providing (1) GIS functionality that is not typical of standard web app development via FOSS4G and (2) a simplified mechanism for installing the software suite. It may seem like a straightforward task to assemble a suite of FOSS4G to address the GIS needs of water resources web apps, especially when there is an abundance of projects to choose from and standardized interfaces provided by OGC. However, to a developer with limited experience, the world of FOSS4G software can quickly become overwhelming and require more time and effort to navigate than most scientists and engineers would be willing to commit—especially for a web app that is secondary to their primary work. Overcoming this hurdle involved over a year of research and careful consideration for the software projects that are included in the Tethys Software Suite (Swain et al., 2015).

The second hurdle overcome by Tethys Software Suite is more subtle. The narrowly focused nature of FOSS4G projects requires several projects to be synthesized to provide all of the typical GIS functionality. Unfortunately, installing many of these projects can be an intensive

process, because the installation instructions are often written for experienced developers. In practice, installing many open source projects requires a great deal of troubleshooting to account for the differences in operating system environments. Tethys Software Suite overcomes this by taking advantage of the Docker virtualization technology.

I developed Docker images for each of the main components of Tethys Software Suite and a command in the Tethys CLI that allows for a single command call to install the entire software suite. Encapsulated in each of the Docker images are weeks to months worth of effort toward installation, troubleshooting, and configuration. Furthermore, the Docker images allow for easy installation on many operating system, which eases deployment of the Tethys Software Suite.

5.2.2 Tethys Software Development Kit

The Tethys SDK overcomes a number of technical hurdles related to app development and interfacing with the Tethys Software Suite. The framework for Tethys apps is based on the Django web framework, but adds an additional layer of abstraction to simplify development. For example, one task that is required to develop web apps is to design the URLs of the app. In a pure Django web app, the URLs are specified using regular expressions. Using the framework provided by Tethys SDK, URLs are specified using strings with a simple syntax for URL variables as illustrated previously in Figure 3-3.

The base template provided for Tethys web apps overcomes a significant hurdle to web app development by eliminating much of the boilerplate code required at the outset of development. In the context of Tethys app development, the boilerplate code includes the HTML that provides unified layout for all the pages in Tethys web apps, the CSS that provides a theme and style for the web app pages, and JavaScript that is used for smooth transitions and other front

end logic. Development of the base template for Tethys apps required several months. For scientists and engineers who are primarily concerned with developing cutting edge workflows and models or collecting and analyzing data, this boilerplate coding for a web app development is often perceived as a distraction to the main objective—to make their work accessible. This results in a poor user experience typical with scientific- or engineering-based web applications and, ironically, does not convey the idea that the user is participating in the latest science or technology. The base template allows developers to immediately start working on the logic and workflow of the app without the need of worrying about how it looks or feels.

Another of the challenges faced by novice water resources web app developers is the multilingual nature of web development. HTML and CSS are required to provide the structure and theme of each page of the web app and JavaScript is used to provide dynamic capabilities such as plotting and mapping. A scripting language such as PHP, Python, or Ruby is also required on the server to handle the logic of the app and interact with the database. The Template Gizmos API greatly reduces the amount of multilingual coding.

For example, the Map View gizmo provides a mechanism for developers to add a dynamic OpenLayers map to a page in their app. The Map View gizmo consists of HTML, CSS and over 1000 lines of JavaScript and represents several months of development. In contrast, Tethys app developers are able to insert a map by creating a configuration object using Python and adding a single line to the HTML template—no JavaScript or CSS required. To be fair, creating maps with advanced features such as clicking on a feature on a map and displaying a plot associated with that feature would require the use of additional JavaScript and CSS, but the amount of coding in multiple languages is significantly reduced by the use of the Template

Gizmos API. Further they provide a set of excellent examples from which a developer can modify or learn from to create new elements of an app.

To illustrate how Tethys Platform addresses the boilerplate code and multilingual coding barriers, I compared the programming language composition of Tethys apps to the composition of the Tethys Django project. The Tethys Django project is used in this comparison used as a benchmark of typical Django web development. Figure 5-1 shows the programming language of composition of Tethys Platform juxtaposed with the language composition of three Tethys apps: Canned GSSHA, GSSHA Index, and Streamflow Prediction Tool.

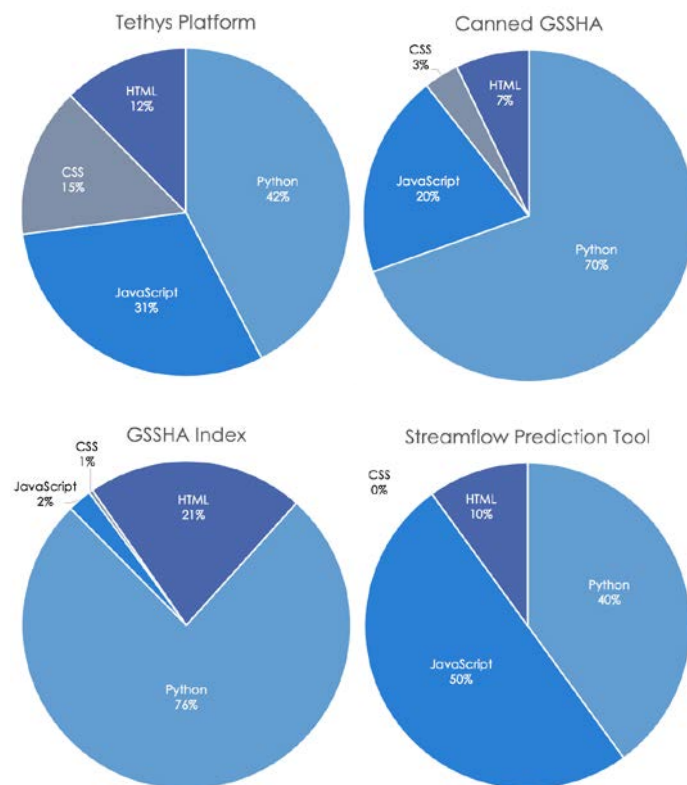


Figure 5-1. Comparison of programming language composition of a normal Django web project (Tethys Platform) and three Tethys apps.

Typical web projects like Tethys Platform consist of a mix of the four web development languages HTML, CSS, JavaScript, and Python. Tethys Platform aims to shift most of the development burden from multiple languages to Python, which is evidenced by the larger proportion of Python in the Canned GSSHA and GSSHA Index plots. The composition of the Streamflow Prediction Tool does not show this trend and is somewhat of an outlier, because the developer of the app preferred more JavaScript development and custom solutions to those provided by Tethys Platform. However, the app also had no CSS styling, because the developer was also not interested in spending time on the styling and user experience of the app. This illustrates how Tethys Platform overcomes the boilerplate code, even for advanced developers.

Another major hurdle cleared by the Tethys SDK is the orchestration and use of the software suite functionality in Tethys apps. This is done by offering a combination of existing third party and custom Python modules. For example, the Persistent Store APIs provide SQLAlchemy and GeoAlchemy as a means for interacting with the PostgreSQL PostGIS-enabled databases of apps. Tethys SDK also provides a command line utility that automates many tasks of database management including creation, updating, and initialization with data. Similarly, the Web Processing Services API provides the OWSLib module as a mechanism for submitting requests to the 52° North WPS using Python.

The Dataset Services API consists of a custom Python module, `tethys_dataset_services`, that includes wrappers for the CKAN, HydroShare, and GeoServer REST APIs. This module facilitates uploading, downloading, and querying file datasets from these services as a means of managing the file datasets. While calls to the REST APIs of this software could be made directly using native Python modules, the `tethys_dataset_services` provides an abstraction layer that simplifies the process.

5.2.3 Tethys Portal

Tethys Portal, the third component of Tethys Platform, overcomes several hurdles related to web app development. First of all, it eliminates the boilerplate code associated with creating a new web site, similar to the manner in which the base template reduces the amount of boilerplate code need to develop apps. Specifically, it provides a user account management system including user profiles that allow users to edit their identifying information and a mechanism for resetting forgotten passwords, authentication and authorization, a homepage, an apps library page that acts as an access point for the installed apps, administrator pages, and web security features. Tethys Platform leverages Django features to provide much of this functionality, but development of these features still required several months. Secondly, Tethys Portal is easily customizable allowing the theme and content to be changed via the admin pages, so that deployed instances can be rebranded to match the organization that hosts it. Tethys Portal makes it significantly easier for organizations to host web apps that they have developed using Tethys Platform.

As an important note, while Tethys Platform seeks to simplify the process of developing web apps for water resources, no restrictions or checks are enforced that would prevent developers with more experience from implementing advanced features. For example, although Tethys provides a simplified mechanism for designing URLs, advanced users are still able to use the regular expressions to gain the benefits of that approach.

6 CONCLUSION

This research has resulted in two primary products that effectively lower the barrier to water resources web app development: (1) a literature review of free and open source software (i.e. software review) and (2) Tethys Platform. The software review included earth science web apps that were published in the peer-reviewed literature in the last decade and it was performed to determine which FOSS4G and FOSS web software has been used to develop such web apps. The FOSS projects presented in the software review do not represent a comprehensive or even representative sampling of all FOSS web GIS and web development software available. Rather, the review highlights 11 FOSS4G software projects and 9 FOSS projects for web development that were used to develop 45 earth sciences web apps. This constitutes a significantly reduced list of possible FOSS software projects that could be used to meet the needs of water resources web app development—greatly lowering the barrier for entry to water resources web development.

The software review includes FOSS4G projects in the categories of spatial databases, spatial data publishing, mapping libraries, and spatial analysis. In the spatial database category, SpatiaLite, MySQL Spatial, and PostGIS were used in earth science web apps. In the spatial publishing category the earth science web apps reviewed used MapServer, GeoServer, and deegree. In the mapping library category, Google Earth™, Google Maps™, and OpenLayers were used. Software used from the spatial analysis category include 52° North WPS, PyWPS, deegree, and GeoServer. The earth science web apps reviewed were developed using a variety

FOSS in the web development category including, Java frameworks, PHP, Perl, the Django Python web framework, the CodeIgniter PHP web framework, Ruby on Rails, Backbone.js, Drupal, and CKAN.

While the software review addresses the hurdle of identifying FOSS software to provide a web framework and spatial data capabilities for water resources web apps, there are still other hurdles that needed to be overcome to make development more viable. Tethys Platform was developed to address these other hurdles and streamline the development of water resources web apps. It consists of three primary components: Tethys Software Suite, Tethys Software Development Kit, and Tethys Portal. Tethys Software Suite includes free and open source software solutions for GIS and distributed computing functionality in water resources web apps. It includes four FOSS4G projects to address the web GIS needs of water resources web apps: the PostGIS extension for PostgreSQL to enable spatial database storage, 52° North Web Processing Service for geoprocessing, GeoServer for generating web friendly maps, and OpenLayers for creating dynamic interactive maps. These software projects implement the Open Geospatial Consortium standards to ensure interoperability. The software suite also includes Google Maps as an alternative for interactive web mapping and HighCharts for dynamic web plotting. Additionally, HTCondor is included to manage computing resources. Tethys Software Development kit provides a framework for developing web apps that includes modules for incorporating the functionality of each software component in the apps. Tethys Portal provides a runtime environment for Tethys web apps and eases the deployment of finished web apps with a website that can be easily rebranded and customized to match the organization hosting it.

Tethys Platform effectively lowers the barrier for water resources web app development by addressing the following hurdles: (1) identify, select, and install software that meet the spatial

and computational capabilities commonly required for water resources modeling; (2) orchestrate the use of multiple FOSS and FOSS4G projects and navigate their differing application programming interfaces (APIs); (3) learn the multi-language programming skills required for modern web development; and (4) develop a web-safe and fully featured web site to host the app. In lowering the barrier, it makes the development of water resources web apps a more viable option for scientists and engineers seeking to make their models and data more accessible. The interactive nature of web apps makes it an excellent medium for conveying complex scientific concepts to lay audiences and creating decision support tools that harness cutting edge modeling techniques. Tethys Platform facilitates making these types of tools more commonplace, which will serve to narrow the gap between research and practice.

REFERENCES

- 52°North. (2014). Home - 52°North Initiative for Geospatial Open Source Software GmbH. Retrieved 12 March 2013, from <http://52north.org/>
- Alconis, J. A., Eco, R. C., Lagmay, A. M. F. A., Aracan, K. A. B., & Seveses, L. J. Z. (2013, December 2013). *The NOAH Initiative: Disaster Management Using WebGIS*. Paper presented at the AGU Fall Meeting, San Fransisco, CA.
- Anderson, J. (2014). *Land use change web app for GSSHA models*. Paper presented at the 2014 Spring Runoff Conference, Utah State University, Logan, Utah, USA.
- Ballatore, A., Tahir, A., McArdle, G., & Bertolotto, M. (2011). A comparison of open source geospatial technologies for web mapping. *International Journal of Web Engineering and Technology*, 6(4), 354-374.
- Bayer, M. (2015). SQLAlchemy: The Database Toolkit for Python. Retrieved 5 May 2015, from <http://www.sqlalchemy.org/>
- Bhuyan, S. J., Koelliker, J. K., Marzen, L. J., & Harrington Jr, J. A. (2003). An integrated approach for water quality assessment of a Kansas watershed. *Environmental Modelling & Software*, 18(5), 473-484. doi: [http://dx.doi.org/10.1016/S1364-8152\(03\)00021-5](http://dx.doi.org/10.1016/S1364-8152(03)00021-5)
- Blodgett, D., Booth, N., Kunicki, T., Walker, J., & Lucido, J. (2012). Description of the US Geological Survey Geo Data Portal Data Integration Framework. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, 5(6), 1687-1691.
- Blower, J. D., Gemmell, A. L., Griffiths, G. H., Haines, K., Santokhee, A., & Yang, X. (2013). A Web Map Service implementation for the visualization of multidimensional gridded environmental data. *Environmental Modelling & Software*, 47(0), 218-224. doi: <http://dx.doi.org/10.1016/j.envsoft.2013.04.002>
- Blower, J. D., Haines, K., Santokhee, A., & Liu, C. L. (2009). GODIVA2: interactive visualization of environmental data on the Web. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1890), 1035-1039.
- Bogdos, N., & Manolakos, E. S. (2013). A tool for simulation and geo-animation of wildfires with fuel editing and hotspot monitoring capabilities. *Environmental Modelling & Software*, 46, 182-195.

- Booth, N. L., Everman, E. J., Kuo, I. L., Sprague, L., & Murphy, L. (2011). A Web- Based Decision Support System for Assessing Regional Water- Quality Conditions and Management Actions1. *JAWRA Journal of the American Water Resources Association*, 47(5), 1136-1150.
- Brooking, C., & Hunter, J. (2013). Providing online access to hydrological model simulations through interactive geospatial animations. *Environmental Modelling & Software*(0). doi: <http://dx.doi.org/10.1016/j.envsoft.2013.01.011>
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-Oriented Software Architectures: A System of Patterns* (Vol. 1): Wiley.
- Carvalho, L. C., Bernardo, S., Orgaz, M. D. M., & Yamazaki, Y. (2010). Forest Fires Mapping and Monitoring of current and past forest fire activity from Meteosat Second Generation Data. *Environmental Modelling & Software*, 25(12), 1909-1914.
- Castronova, A. M., Goodall, J. L., & Elag, M. M. (2013). Models as web services using the Open Geospatial Consortium (OGC) Web Processing Service (WPS) standard. *Environmental Modelling & Software*, 41, 72-83. doi: Doi 10.1016/J.Envsoft.2012.11.010
- Cau, P., Manca, S., Soru, C., Muroli, D., Gorgan, D., Bacu, V., . . . Giuliani, G. (2013). An Interoperable, GIS-oriented, Information and Support System for Water Resources Management. *International Journal of Advanced Computer Science and Applications*, 3(3), 75-82. doi: 10.14569/SpecialIssue.2013.030309
- Cechini, M., Murphy, K., Boller, R., Schmaltz, J., Thompson, C., Huang, T., . . . Roberts, J. (2013). *Expanding Access and Usage of NASA Near Real-Time Imagery and Data*. Paper presented at the AGU Fall Meeting Abstracts.
- Chambers, J. (2013). The R Project for Statistical Computing. from <http://www.r-project.org/>
- Chambers, J. (2013). The R Project for Statistical Computing. Retrieved 12 March 2013, from <http://www.r-project.org/>
- Chen, D., Shams, S., Carmona-Moreno, C., & Leone, A. (2010). Assessment of open source GIS software for water resources management in developing countries. *Journal of Hydro-environment Research*, 4(3), 253-264. doi: 10.1016/j.jher.2010.04.017
- Choi, J., Engel, B., & Farnsworth, R. (2005). Web-based GIS and spatial decision support system for watershed management. *Journal of Hydroinformatics*, 7, 165-174.
- Choi, J., Engel, B., Theller, L., & Harbor, J. (2005). Utilizing web-based GIS and SDSS for hydrological land use change impact assessment. *Transactions of the ASAE*, 48(2), 815-822.
- CKAN. (2013). CKAN, the world's leading open-source data portal platform. 2013, from <http://ckan.org/>

- David, C. H., Maidment, D. R., Niu, G.-Y., Yang, Z.-L., Habets, F., & Eijkhout, V. (2011). River network routing on the NHDPlus dataset. *Journal of Hydrometeorology*, 12(5), 913-934.
- Davies, D. K., Ilavajhala, S., Wong, M. M., & Justice, C. O. (2009). Fire information for resource management system: archiving and distributing MODIS active fire data. *Geoscience and Remote Sensing, IEEE Transactions on*, 47(1), 72-79.
- de Sousa, L., Eykamp, C., Leopold, U., Baume, O., & Braun, C. (2012). *iGUESS-A web based system integrating urban energy planning and assessment modelling for multi-scale spatial decision making*. Paper presented at the International Congress on Environmental Modelling and Software Managing Resources of a Limited Planet, Sixth Biennial Meeting. Leipzig, Germany. R. Seppelt, AA Voinov, S. Lange, D. Bankamp (Eds.).
- Delipetrev, B., Jonoski, A., & Solomatine, D. (2012). *Cloud Computing Framework for a Hydro Information System*. Faculty of Computer Science, University of Goce Delcev, Stip, Republic of Macedonia.
- Delipetrev, B., Jonoski, A., & Solomatine, D. P. (2014). Development of a web application for water resources based on open source software. *Computers & Geosciences*, 62, 35-42.
- Demir, I., & Krajewski, W. F. (2013). Towards an integrated Flood Information System: Centralized data access, analysis, and visualization. *Environmental Modelling & Software*, 50, 77-84.
- DeVantier, B. A., & Feldman, A. D. (1993). Review of GIS Application in Hydrologic Modeling. *Journal of Water Resources Planning and Management*, 119(2), 1 - 13.
- Díaz, L., Granell, C., & Gould, M. (2008). Case Study: Geospatial Processing Services for Web based Hydrological Applications: Springer.
- Django Software Foundation. (2015). The web framework for perfectionists with deadlines - Django. Retrieved 11 May 2015, from <https://www.djangoproject.com/>
- Docker Inc. (2015). Docker: Build, Ship, and Run Any App, Anywhere. from <https://www.docker.com/>
- Dolder, H., Jones, N., & Nelson, E. J. (2015). Simple Method for Using Precomputed Hydrologic Models in Flood Forecasting with Uniform Rainfall and Soil Moisture Pattern. *Journal of Hydrologic Engineering*, 0(0), 04015039. doi: doi:10.1061/(ASCE)HE.1943-5584.0001232
- Downer, C. W., & Ogden, F. L. (2004). GSSHA: Model To Simulate Diverse Stream Flow Producing Processes. *Journal of Hydrologic Engineering*, 9(3), 161-174. doi: 10.1061/(asce)1084-0699(2004)9:3(161)
- Drupal. (2013). Drupal. 2013, from <https://drupal.org/>

- Dubois, G., Schulz, M., Skøien, J., Bastin, L., & Peedell, S. (2013). eHabitat, a multi-purpose Web Processing Service for ecological modeling. *Environmental Modelling & Software*, 41, 123-133.
- ESRI. (2004). ArcGIS Server: ESRI's Enterprise GIS Application Server. http://downloads.esri.com/support/whitepapers/other/_arcgis-server_90.pdf
- Fang, S., Xu, L., Zhu, Y., Liu, Y., Liu, Z., Pei, H., . . . Zhang, H. (2013). An integrated information system for snowmelt flood early-warning based on internet of things. *Information Systems Frontiers*, 1-15.
- Fang, Y., & Feng, M. (2009). A WebGIS framework for vector geospatial data sharing based on open source projects.
- Feng, M., Liu, S., Euliss Jr, N. H., Young, C., & Mushet, D. M. (2011). Prototyping an online wetland ecosystem services model using open model sharing standards. *Environmental Modelling & Software*, 26(4), 458-468. doi: <http://dx.doi.org/10.1016/j.envsoft.2010.10.008>
- Flaishans, J., Hong, T., Snyder, M., Pascale, C., & Purucker, S. (2014). *Fronting Integrated Scientific Web Applications: Design Features and Benefits for Regulatory Environments*. Paper presented at the International Environmental Modelling and Software Society (iEMSs) 7th International Congress on Environmental Modelling and Software, San Diego.
- Frehner, M., & Brändli, M. (2006). Virtual database: Spatial analysis in a Web-based data management system for distributed ecological data. *Environmental Modelling & Software*, 21(11), 1544-1554. doi: <http://dx.doi.org/10.1016/j.envsoft.2006.05.012>
- Furieri, A. (2008). SpatiaLite-A complete spatial DBMS in a nutshell.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison-Wesley.
- GeoServer. (2013). GeoServer Documentation. 2013, from <http://docs.geoserver.org/>
- GeoTools. (2014). GeoTools - The Open Source Java GIS Toolkit. Retrieved 23 October 2014, from <http://geotools.org/>
- Gkatzoflias, D., Mellios, G., & Samaras, Z. (2013). Development of a web GIS application for emissions inventory spatial allocation based on open source software tools. *Computers & Geosciences*, 52(0), 21-33. doi: <http://dx.doi.org/10.1016/j.cageo.2012.10.011>
- Goharian, E., Burian, S., Bardsley, T., & Strong, C. (2015). A Vulnerability Assessment to Integrate Flooding and Shortage of Water Supply Systems under Climate Change Conditions. *Journal of Water Resources Planning and Management*.

- Goharian, E., Burian, S., & Karamouz, M. (2015). Using Reliability-Vulnerability Joint Probability Distribution to Evaluate the Performance of Water Supply System Performance. *Water Resources Management*.
- Goharian, E., & Burian, S. J. (2014). *Integrated Urban Water Resources Modeling In A Semi-Arid Mountainous Region Using A Cyber-Infrastructure Framework*. Paper presented at the HIC2014 - 11th International Conference on Hydroinformatics, New York City, New York, USA.
- GoldSim Technology Group. (2015). Monte Carlo Simulation Software - GoldSim. Retrieved 5 May 2015, from <http://www.goldsim.com/Home/>
- Goodrich, D. C., Guertin, D. P., Burns, I. S., Nearing, M. A., Stone, J. J., Wei, H., . . . Pierson, F. (2008). RHEM Web Tool: Rangeland Hydrology Erosion Model Web Tool. *Rangelands*. Retrieved February 28, 2014, 2014, from <http://apps.tucson.ars.ag.gov/rhem/>
- Goodrich, D. C., Guertin, D. P., Burns, I. S., Nearing, M. A., Stone, J. J., Wei, H., . . . Pierson, F. (2011). AGWA: the automated geospatial watershed assessment tool to inform rangeland management. *Rangelands*, 33(4), 41-47.
- Google. (2014a). Announcing deprecation of the Google Earth API. Retrieved January 9, 2015, from <http://googlegeodevelopers.blogspot.com.au/2014/12/announcing-deprecation-of-google-earth.html>
- Google. (2014b). Google Maps JavaScript API v3. Retrieved 12 June 2014, from <https://developers.google.com/maps/web/>
- Granell, C., Díaz, L., & Gould, M. (2010). Service-oriented applications for environmental models: Reusable geospatial services. *Environmental Modelling & Software*, 25(2), 182-198.
- GRASS Development Team. (2014). GRASS GIS: The world's leading Free GIS software. Retrieved 23 October 2014, from <http://grass.osgeo.org/download/>
- Guzzetti, F., & Tonelli, G. (2004). Information system on hydrological and geomorphological catastrophes in Italy (SICI): a tool for managing landslide and flood hazards. *Natural Hazards and Earth System Science*, 4(2), 213-232.
- Hall, D. K., V. V. Salomonson, and G. A. Riggs. 2006. *MODIS/Terra Snow Cover Daily L3 Global 500m Grid*. Version 5. MOD10A1. Boulder, Colorado USA: NASA National Snow and Ice Data Center Distributed Active Archive Center. <http://dx.doi.org/10.5067/63NQASRPDB0>.
- Han, W., Di, L., Zhao, P., & Shao, Y. (2012). DEM Explorer: An online interoperable DEM data sharing and analysis system. *Environmental Modelling & Software*, 38, 101-107.
- Hazzard, E. (2011). *OpenLayers 2.10 : beginner's guide*. Birmingham: Packt Publishing.

- Highsoft AS. (2014). Highcharts - Interactive JavaScript charts for your web projects. 2014, from <http://www.highcharts.com/>
- Hill, D. J., Liu, Y., Marini, L., Kooper, R., Rodriguez, A., Futrelle, J., . . . McLaren, T. (2011). A virtual sensor system for user-generated, real-time environmental data products. *Environmental Modelling & Software*, 26(12), 1710-1724.
- Holl, S., & Plum, H. (2009). PostGIS. *GeoInformatics*, 03/2009, 34-36. doi: citeulike-article-id:4463470
- Horsburgh, J. S., Tarboton, D. G., Schreuders, K. A., Maidment, D. R., Zaslavsky, I., & Valentine, D. (2010). Hydroserver: A platform for publishing space-time hydrologic datasets.
- Iacovella, S., & Youngblood, B. (2013). *GeoServer Beginner's Guide*: Packt Publishing.
- Iwanaga, T., El Sawah, S., & Jakeman, A. (2013). Design and implementation of a Web-based groundwater data management system. *Mathematics and Computers in Simulation*, 93, 164-174.
- Jansson, P.-E., & Moon, D. S. (2001). A coupled model of water, heat and mass transfer using object orientation to improve flexibility and functionality. *Environmental Modelling & Software*, 16(1), 37-46. doi: [http://dx.doi.org/10.1016/S1364-8152\(00\)00062-1](http://dx.doi.org/10.1016/S1364-8152(00)00062-1)
- Jones, N., Nelson, J., Swain, N., Christensen, S., Tarboton, D., & Dash, P. (2014). *Tethys: A Software Framework for Web-Based Modeling and Decision Support Applications*. Paper presented at the International Environmental Modelling and Software Society (iEMSS) 7th International Congress on Environmental Modelling and Software, San Diego, California, USA.
- Jung, J. K., Sinha, S. K., & Whittle, L. G. (2013). Development of a Water Infrastructure Knowledge Database. *Journal of Infrastructure Systems*.
- Kralidis, T. (2010). OWSLib Documentation. Retrieved 5 May 2015, from <http://geopython.github.io/OWSLib/>
- Kulkarni, A., Mohanty, J., Eldho, T., Rao, E., & Mohan, B. (2014). A web GIS based integrated flood assessment modeling tool for coastal urban watersheds. *Computers & Geosciences*, 64, 7-14.
- Lam, D., Leon, L., Hamilton, S., Crookshank, N., Bonin, D., & Swayne, D. (2004). Multi-model integration in a decision support system: a technical user interface approach for watershed and lake management scenarios. *Environmental Modelling & Software*, 19(3), 317-324. doi: [http://dx.doi.org/10.1016/S1364-8152\(03\)00156-7](http://dx.doi.org/10.1016/S1364-8152(03)00156-7)
- Lemoine, E. (2015). GeoAlchemy 2 Documentation. Retrieved 5 May 2015, from <http://geoalchemy-2.readthedocs.org/en/0.2.4/>

- Leone, A., Shams, S., & Chen, D. (2006). An object- oriented and OpenGIS supported hydro information system (3O- HIS) for upper Mersey river basin management. *International Journal of River Basin Management*, 4(2), 99-107.
- Li, S., Saborowski, J., Nieschulze, J., Li, Z.-y., Lu, Y.-c., & Chen, E.-x. (2007). Web service based spatial forest information system using an open source software approach. *Journal of Forestry Research*, 18(2), 85-90. doi: 10.1007/s11676-007-0017-9
- Li, Y., Fang, X., & Jiao, S. (2013). A spatial decision support system for water resource management of Yellow River Basin in China. *Hydraulic Engineering*, 83.
- Lim, K. J., Engel, B. A., Tang, Z., Choi, J., Kim, K. S., Muthukrishnan, S., & Tripathy, D. (2005). Automated web gis based hydrograph analysis tool, WHAT1: Wiley Online Library.
- Litzkow, M. J., Livny, M., & Mutka, M. W. (1988, 13-17 Jun 1988). *Condor-a hunter of idle workstations*. Paper presented at the Distributed Computing Systems, 1988., 8th International Conference on.
- Livny, M., Basney, J., Raman, R., & Tannenbaum, T. (1997). Mechanisms for high throughput computing. *SPEEDUP journal*, 11(1), 36-40.
- Maidment, D. (2014). *National Flood Interoperability Experiment*. Paper presented at the AGU Fall Meeting Abstracts.
- Maidment, D., Helly, J., Kumar, P., Piasecki, M., & Hooper, R. (2004). *CUAHSI hydrologic information system*. Paper presented at the AGU Fall Meeting Abstracts.
- Maidment, D. R. (2014). *National Flood Interoperability Experiment*. Paper presented at the AGU Fall Meeting Abstracts.
- Maidment, D. R. (2015). *A Conceptual Framework for the National Flood Interoperability Experiment*. White Paper. Center for Research in Water Resources. University of Texas at Austin. Austin Texas.
- Mason, S. J. K., Cleveland, S. B., Llovet, P., Izurieta, C., & Poole, G. C. (2014). A centralized tool for managing, archiving, and serving point-in-time data in ecological research laboratories. *Environmental Modelling & Software*, 51(0), 59-69. doi: <http://dx.doi.org/10.1016/j.envsoft.2013.09.008>
- Melis, M. T., Locci, F., Dessì, F., Frigerio, I., Strigaro, D., & Vuillermoz, E. (2014). *SHARE Geonetwork, a system for climate and paleoclimate data sharing*. Paper presented at the Proceedings of the 7th International Congress on Environmental Modelling and Software (iEMSs).
- Miller, S. N., Semmens, D. J., Goodrich, D. C., Hernandez, M., Miller, R. C., Kepner, W. G., & Guertin, D. P. (2007). The Automated Geospatial Watershed Assessment tool.

- Environmental Modelling & Software*, 22(3), 365-377. doi:
<http://dx.doi.org/10.1016/j.envsoft.2005.12.004>
- Millman, K. J., & Aivazis, M. (2011). Python for scientists and engineers. *Computing in Science & Engineering*, 13(2), 9-12.
- Müller, M. (2007). deegree-Building Blocks for Spatial Data. *OSGeo Journal*, 1(1).
- Nguyen, T. T. (2009a). Indexing PostGIS Databases and Spatial Query Performance Evaluations. *International Journal of Geoinformatics*, 5(3), 1-9.
- Nguyen, T. T. (2009b). Indexing PostGIS Databases and Spatial Query Performance Evaluations *International Journal of Geoinformatics* (Vol. 5, pp. 1-9).
- OGC. (2006). OpenGIS Web Map Server Implementation Specification.
- OGC. (2007a). Catalogue Services Specification.
- OGC. (2007b). OpenGIS Web Processing Service.
- OGC. (2008). OGC KML.
- OGC. (2010a). Open GIS Web Map Tile Service Implementation Standard. Retrieved 23 October 2014, from <http://www.opengeospatial.org/standards/wmts>
- OGC. (2010b). OpenGIS Implementation Standard for Geographic information - Simple Feature access - Part 2: SQL Option.
- OGC. (2012a). Geospatial and location standards for: 2012, from <http://www.opengeospatial.org/>
- OGC. (2012b). OGC Geography Markup Language (GML) - Extended schemas and encoding rules.
- OGC. (2012c). OGC WCS 2.0 Interface Standard - Core: Corrigendum.
- OGC. (2014). OGC Web Feature Service 2.0 Interface Standard.
- Olaya, V., & Gimenez, J. C. (2011). *SEXTANTE, a versatile open-source library for spatial data analysis*.
- Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science & Engineering*, 9(3), 10-20.
- Open Geospatial Consortium. (2015). Implementing Products. Retrieved 11 May 2015, from <http://www.opengeospatial.org/resource/products>
- Open Knowledge Foundation. (2014). ckan - The open source data portal software. 2014, from <http://ckan.org/>

- Open Source Geospatial Foundation. (2014). GEOS - Geometry Engine Open Source. Retrieved 23 October 2014, from <http://trac.osgeo.org/geos/>
- Oracle. (2012). Your First Cup. Retrieved 28 October 2014, from <http://docs.oracle.com/javase/6/firstcup/doc/gkhoy.html>
- OSGeo. (2014). Benchmarking 2011. Retrieved January 9, 2015, from http://wiki.osgeo.org/wiki/Benchmarking_2011
- Oulidi, H. J., Löwner, R., Benaabidate, L., & Wächter, J. (2012). HydrIS: An open source GIS decision support system for groundwater management (Morocco). *Geo-spatial Information Science*, 12(3), 212-216.
- Peres, A., Miletto, E. M., Kapusta, S., Ojeda, T., Lacasse, A., & Gagnon, J. (2013). WAITS—AN IT STRUCTURE FOR ENVIRONMENTAL INFORMATION VIA OPEN KNOWLEDGE, DYNAMIC DASHBORARDS AND SOCIAL WEB OF THINGS.
- Python Software Foundation. (2013). Python. from <http://python.org/about/>
- Rao, M., Fan, G., Thomas, J., Cherian, G., Chudiwale, V., & Awawdeh, M. (2007). A web-based GIS Decision Support System for managing and planning USDA's Conservation Reserve Program (CRP). *Environmental Modelling & Software*, 22(9), 1270-1280.
- Rojas-Sola, J. I., Castro-García, M., & Carranza-Cañadas, M. d. P. (2011). Content management system incorporated in a virtual museum hosting. *Journal of Cultural Heritage*, 12(1), 74-81. doi: <http://dx.doi.org/10.1016/j.culher.2010.10.004>
- Royappa, A. V. (2000). The PHP web application server. *Journal of Computing Sciences in Colleges*, 15(3), 201-211.
- Safe Software. (2014). FME Products. Retrieved 23 October 2014, from <http://www.safe.com/fme/>
- Saguibo, M. (2015). *Observed Data App using the Tethys Platform*. (Master of Science Project), Brigham Young University, Provo, Utah, USA.
- Sanner, M. F. (1999). Python: a programming language for software integration and development. *J Mol Graph Model*, 17(1), 57-61.
- Santhi, C., Srinivasan, R., Arnold, J. G., & Williams, J. R. (2006). A modeling approach to evaluate the impacts of water quality management plans implemented in a watershed in Texas. *Environmental Modelling & Software*, 21(8), 1141-1157. doi: <http://dx.doi.org/10.1016/j.envsoft.2005.05.013>
- Schut, P. (2007). OpenGIS Web Processing Service. (pp. 87): Open Geospatial Consortium Inc., Wayland, MA, USA.

- Simao, A., Densham, P. J., & Haklay, M. (2009). Web-based GIS for collaborative planning and public participation: An application to the strategic planning of wind farm sites. *Journal of Environmental Management*, 90(6), 2027-2040.
- Singh, P. S., Chutia, D., & Sudhakar, S. (2012). Development of a Web Based GIS Application for Spatial Natural Resources Information System Using Effective Open Source Software and Standards. *Journal of Geographic Information System*, 4(3), 261-266. doi: 10.4236/jgis.2012.43031
- Snow, A. (2015). *A New Global Forecasting Model to Produce High-Resolution Stream Forecasts*. (Master of Science), Brigham Young University, Provo, Utah, USA.
- Steiniger, S., & Hunter, A. J. S. (2012a). Free and Open Source GIS Software for Building a Spatial Data Infrastructure. In E. Bocher & M. Neteler (Eds.), *Geospatial Free and Open Source Software in the 21st Century* (pp. 247-261): Springer Berlin Heidelberg.
- Steiniger, S., & Hunter, A. J. S. (2012b). Review: The 2012 free and open source GIS software map – A guide to facilitate research, development, and adoption. *Computers, Environment and Urban Systems*. doi: 10.1016/j.compenvurbsys.2012.10.003
- Steiniger, S., & Weibel, R. (2010). GIS Software: a description in 1000 word *Encyclopedia of Geography*. London CB: University of Zurich - Zurich Open Repository and Archive.
- Sugrue, J. (2013). *Beginning Backbone.js*: Apress.
- Sun, A. (2013). Enabling collaborative decision-making in watershed management using cloud-computing services. *Environmental Modelling & Software*, 41, 93-97.
- Sun, X., Shen, S., Leptoukh, G. G., Wang, P., Di, L., & Lu, M. (2012). Development of a Web-based visualization platform for climate research using Google Earth. *Computers & Geosciences*, 47(0), 160-168. doi: 10.1016/j.cageo.2011.09.010
- Swain, N. R., Latu, K., Christensen, S. D., Jones, N. L., Nelson, E. J., Ames, D. P., & Williams, G. P. (2015). A review of open source software solutions for developing water resources web applications. *Environmental Modelling & Software*, 67(0), 108-117. doi: <http://dx.doi.org/10.1016/j.envsoft.2015.01.014>
- Tarboton, D., Idaszak, R., Horsburgh, J., Heard, J., Ames, D., Goodall, J., . . . Arrigo, J. (2014). *HydroShare: Advancing Collaboration through Hydrologic Data and Model Sharing*. Paper presented at the International Environmental Modelling and Software Society (iEMSS) 7th International Congress on Environmental Modelling and Software San Diego, California, USA, DP Ames, N. Quinn(Eds.) <http://www.iemss.org/society/index.php/iemss-2014-proceedings>.
- Tate, B. A., & Hibbs, C. (2006). *Ruby on Rails: Up and Running: Up and Running*: " O'Reilly Media, Inc."

- The Python Wiki. (2014). WebFrameworks - Python Wiki. Retrieved April 9, 2014, from <https://wiki.python.org/moin/WebFrameworks>
- Thompson, C., Cechini, M., Huang, T., Roberts, J., Alarcon, C., Boller, R., . . . Schmaltz, J. (2014). *Gibs: A Rich Visual Interface to NASA's Earth Science Data Holdings*. Paper presented at the AGU Fall Meeting Abstracts.
- Upton, D. (2007). *CodeIgniter for Rapid PHP Application Development*. Packt Publishing Ltd.
- USDA Forest Service. (2015). Uinta-Wasatch-Cache National Forest - Resource Management. Retrieved 20 May 2015, from http://www.fs.usda.gov/detailfull/uwcnf/landmanagement/resourcemanagement/?cid=fse_m_035491&width=full
- Valentine, D., Taylor, P., & Zaslavsky, I. (2012). *WaterML, an information standard for the exchange of in-situ hydrological observations*. Paper presented at the EGU General Assembly Conference Abstracts.
- Van Knowe, G., Waight, K., Barlow, P., Yalda, S., Zoppetti, G., Johnson, R., . . . Tang, B. (2014). Creating an On-Line Interactive Earth Science Environmental Simulator.
- Vatsavai, R., Shekhar, S., Burk, T., & Lime, S. (2006). UMN-MapServer: A High-Performance, Interoperable, and Open Source Web Mapping and Geo-spatial Analysis System Geographic Information Science. In M. Raubal, H. Miller, A. Frank & M. Goodchild (Eds.), (Vol. 4197, pp. 400-417): Springer Berlin / Heidelberg.
- Vivid Solutions. (2014). JTS Topology Suite. Retrieved 23 October 2014, from <http://www.vividsolutions.com/jts/JTSHome.htm>
- Walker, J. D., & Chapra, S. C. (2014). A client-side web application for interactive environmental simulation modeling. *Environmental Modelling & Software*, 55(0), 49-60. doi: <http://dx.doi.org/10.1016/j.envsoft.2014.01.023>
- Walker, J. D., & Chapra, S. C. (2014). A client-side web application for interactive environmental simulation modeling. *Environmental Modelling & Software*, 55, 49-60.
- Wan, Z., Hong, Y., Khan, S., Gourley, J., Flamig, Z., Kirschbaum, D., & Tang, G. (2014). A cloud-based global flood disaster community cyber-infrastructure: Development and demonstration. *Environmental Modelling & Software*, 58, 86-94.
- Warmerdam, F. (2008). The geospatial data abstraction library *Open Source Approaches in Spatial Data Handling* (pp. 87-104): Springer.
- Zhao, P., Foerster, T., & Yue, P. (2012). The Geoprocessing Web. *Computers & Geosciences*, 47(0), 3-12. doi: 10.1016/j.cageo.2012.04.021

APPENDIX A. SUPPLEMENTARY PYTHON LIBRARIES

This appendix includes descriptions of MapKit and GsshaPy, two Python modules that I developed as part of CI-WATER but do not necessarily fall under the umbrella of Tethys Platform. Both modules were used as part of the workflow for the GSSHA Index Map Editor app, but are general in nature to be useful for any app developers creating web applications that use the GSSHA model.

A.1 MapKit

MapKit is a general purpose Python library that can be used to generate KML visualizations from the spatial fields of PostGIS databases (Figure A-1). The module provides objects and functions that can be used to load rasters of various types into a PostGIS database table with raster fields. It also provides functions that employ the capabilities of the built-in PostGIS geoprocessing functions to export PostGIS raster or geometry fields as KML, GeoJSON, or Well Known Text. MapKit includes utility functions for generating color ramps that are applied to the KML maps and supplies several default color ramps for conveniences. MapKit is distributed on the Python Package Index and can be installed using pip or easy_install.

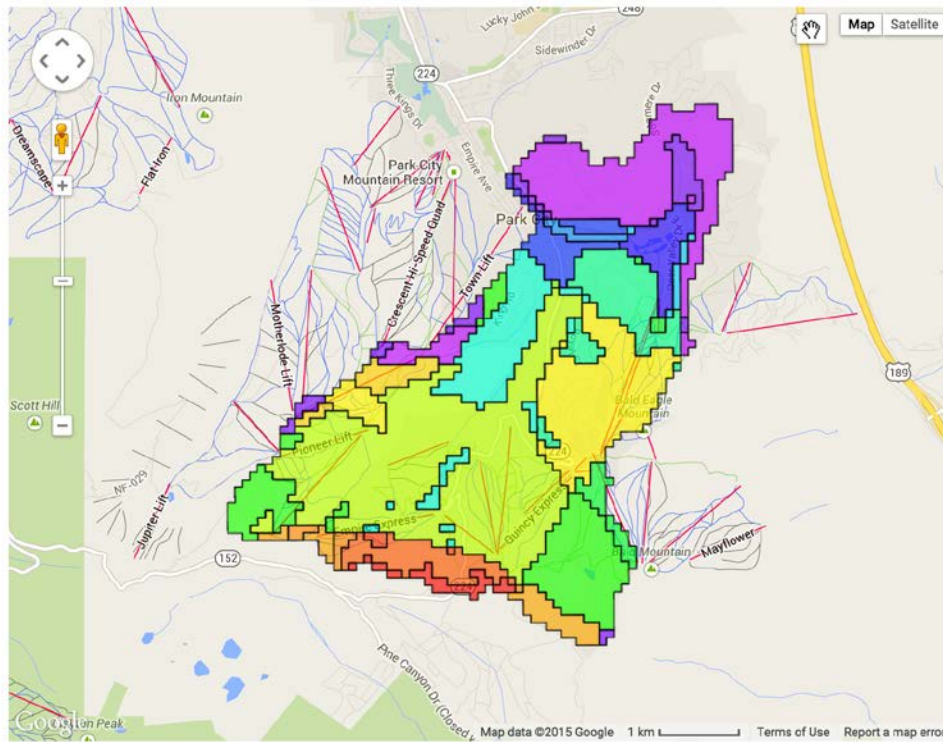


Figure A-1. KML visualization of a GSSHA index map generated with MapKit.

A.2 GsshaPy

GsshaPy is an object relational model (ORM) for GSSHA models that was built to provide a simple mechanism for working with these models in a web environment. GsshaPy is built on the SQLAlchemy ORM, which provides an object-oriented interface for SQL databases. With SQLAlchemy tables of the database can be defined with classes and instances of those classes will represent individual records in the table. GsshaPy provides SQLAlchemy classes that define the tables of a data model that can be used to store a majority of the data from GSSHA models. The tables for new GsshaPy databases can be created with only a few lines of Python code and the classes can be used to query the database.

In addition to providing an ORM and data model for GSSHA files, GsshaPy also provides file parsers and file writers that automate the process of loading data into the database and writing it out to file. Once in a database, the files can be manipulated using the Python ORM and easily accessed for display on web pages. After editing is complete, the contents of any GsshaPy database can be written back to file to allow model execution.

GsshaPy is also capable of storing the spatial model files in spatial fields of a PostGIS enabled PostgreSQL database. This exposes the GSSHA spatial files to the geoprocessing database functions provided by PostGIS. GsshaPy leverages these functions coupled with MapKit to generate KML visualizations of the spatial files of GSSHA including the index maps, raster maps, and stream network. GsshaPy is distributed on the Python Package Index and can be installed using pip or easy_install. The entire process is illustrated in Figure A-2.

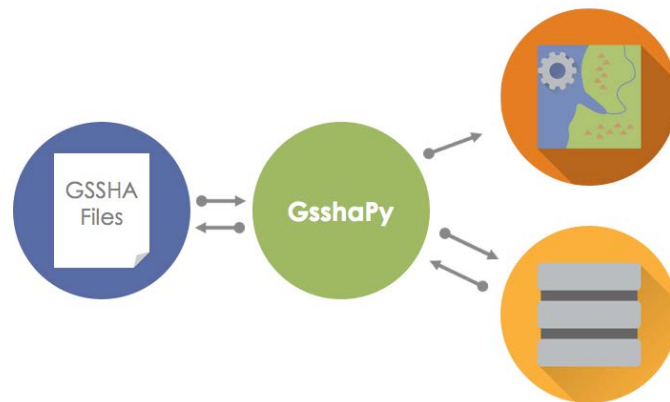


Figure A-2. GsshaPy can be used to interface between GSSHA model files and an SQL database.

APPENDIX B. SOFTWARE AVAILABILITY

Tethys Platform is free and open source software available under the BSD 2-Clause license and is made publicly available through several GitHub repositories (Table B-1). The Dockerfiles and supporting materials that are used to build the Docker images for some of the software suite components are located in a separate repository to prevent the primary Tethys repository from becoming too large. The Python interfaces for GeoServer, CKAN, and HydroShare that make up the Dataset Services API are maintained in a separate repository and updates are released on the Python Package Index (PyPI) independent of Tethys Platform. This is done to allow the interfaces to be used in Python scripts that are external to Tethys Platform.

Table B-1. Primary Tethys Platform Source Code Repositories

Description	Repository
Primary Tethys Repository	https://github.com/CI-WATER/tethys
Tethys Docker Files	https://github.com/CI-WATER/tethys_docker
Dataset Services Python Interfaces	https://github.com/CI-WATER/tethys_datasets_services

The source code for the example Tethys apps that were described in this work are also available in GitHub repositories (Table B-2). The ECMWF-RAPI Flood Prediction Tool includes several repositories: one for the Tethys app and several others to support the external web processing

services. The source code for MapKit and GsshaPy is also available in GitHub repositories (Table B-3).

Table B-2. Tethys App Source Code Repositories

Tethys App	Source Code Repositories
Canned GSSHA	https://github.com/CI-WATER/tethysapp-canned_gssha
Parely’s Creek Management Tool	https://github.com/CI-WATER/tethysapp-parleys_creek_management
GSSHA Index Map Editor	https://github.com/CI-WATER/tethysapp-gsshaindex
Streamflow Prediction Tool	https://github.com/CI-WATER/tethysapp-erfp_tool https://github.com/CI-WATER/sfpt_dataset_manager https://github.com/CI-WATER/erfp_data_process_ubuntu_aws https://github.com/CI-WATER/erfp_data_process_ubuntu
Observed Data	https://github.com/CI-WATER/tethysapp-observed_data
Snow Inspector	https://github.com/jirikadlec2/snow-inspector
Shapefile Viewer	https://github.com/SarvaPulla/sfviewer
HIS Time Series Viewer	https://github.com/zhiyuli/tethysapp-refts_viewer
Raster Viewer	https://github.com/zhiyuli/tethysapp-raster_viewer

Table B-3. MapKit and GsshaPy Source Code Repositories

Project	Repository
MapKit	https://github.com/CI-WATER/mapkit
GsshaPy	https://github.com/CI-WATER/gsshapy

Extensive documentation for Tethys Platform, including installation instructions, tutorials, and detailed examples of SDK usage is available at: <http://docs.tethys.ci-water.org>. For more information about Tethys Platform, please visit <http://tethys.ci-water.org>.

APPENDIX C. TETHYS PLATFORM DOCUMENTATION

The following appendix includes the documentation that was written for Tethys Platform. The documentation includes explanations of the major components of Tethys Platform, code examples, and tutorials to orient new Tethys developers. The documentation is available on the Internet at <http://docs.tethys.ci-water.org>. The website automatically generates a PDF version of the documentation for download, which is the version that is included in this appendix. Note that the page numbering and formatting in this appendix are the native page numbering and formatting of the PDF document.

Tethys Platform Documentation

Release 1.1.0

Nathan Swain

May 29, 2015

1	Contents	3
1.1	Features	3
1.2	What's New	9
1.3	Installation	10
1.4	Tutorials	29
1.5	Software Suite	55
1.6	Software Development Kit	62
1.7	Tethys Portal	171
1.8	Deploy	177
1.9	Source Code	187
1.10	Develop Tethys Platform	187
1.11	Supplementary	192
1.12	Summary of References	200
1.13	Glossary	205
2	Acknowledgements	207
3	Indices and tables	209



Last Updated: May 28, 2015


Tethys is a platform that can be used to develop and host water resources web applications or web apps. It includes a suite of free and open source software (FOSS) that has been carefully selected to address the unique development needs of water resources web apps. Tethys web apps are developed using a Python software development kit (SDK) which includes programmatic links to each software component. Tethys Platform is powered by the Django Python web framework giving it a solid web foundation with excellent security and performance. Refer to the [Features](#) article for an overview of the features of Tethys Platform.

Important: We are pleased to announce that Tethys Platform 1.1.0 has arrived! Check out the [What's New](#) article for a description of the new features and changes.

1.1 Features

Last Updated: May 28, 2015

Tethys is a platform that can be used to develop and host engaging, interactive water resources web applications or web apps. It includes a suite of free and open source software (FOSS) that has been carefully selected to address the unique development needs of water resources web apps. Tethys web apps are developed using a Python software development kit (SDK) which includes programmatic links to each software component. Tethys Platform is powered by the [Django](#) Python web framework giving it a solid web foundation with excellent security and performance.



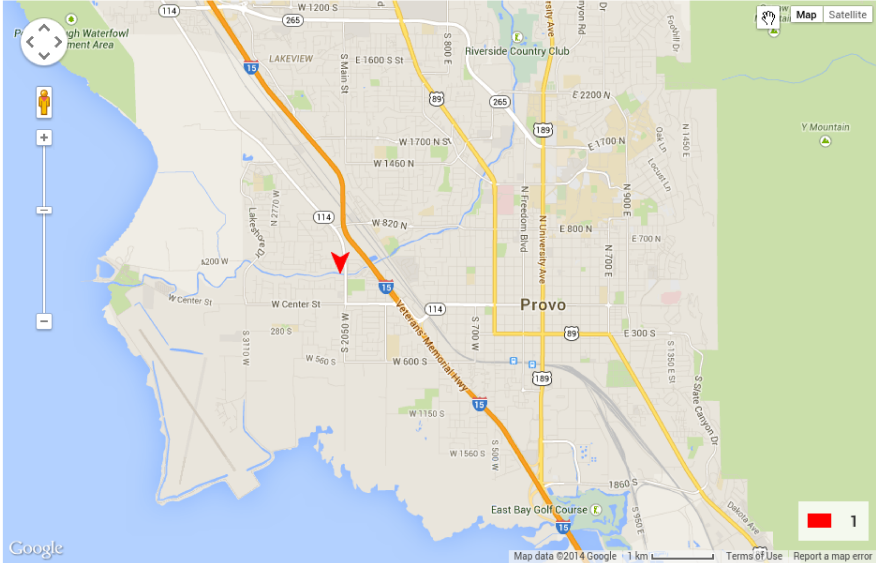
Stream Gages

Exit

Gages

- All Gages
- Stream Gage 1**
- Stream Gage 2
- Stream Gage 3
- Stream Gage 4

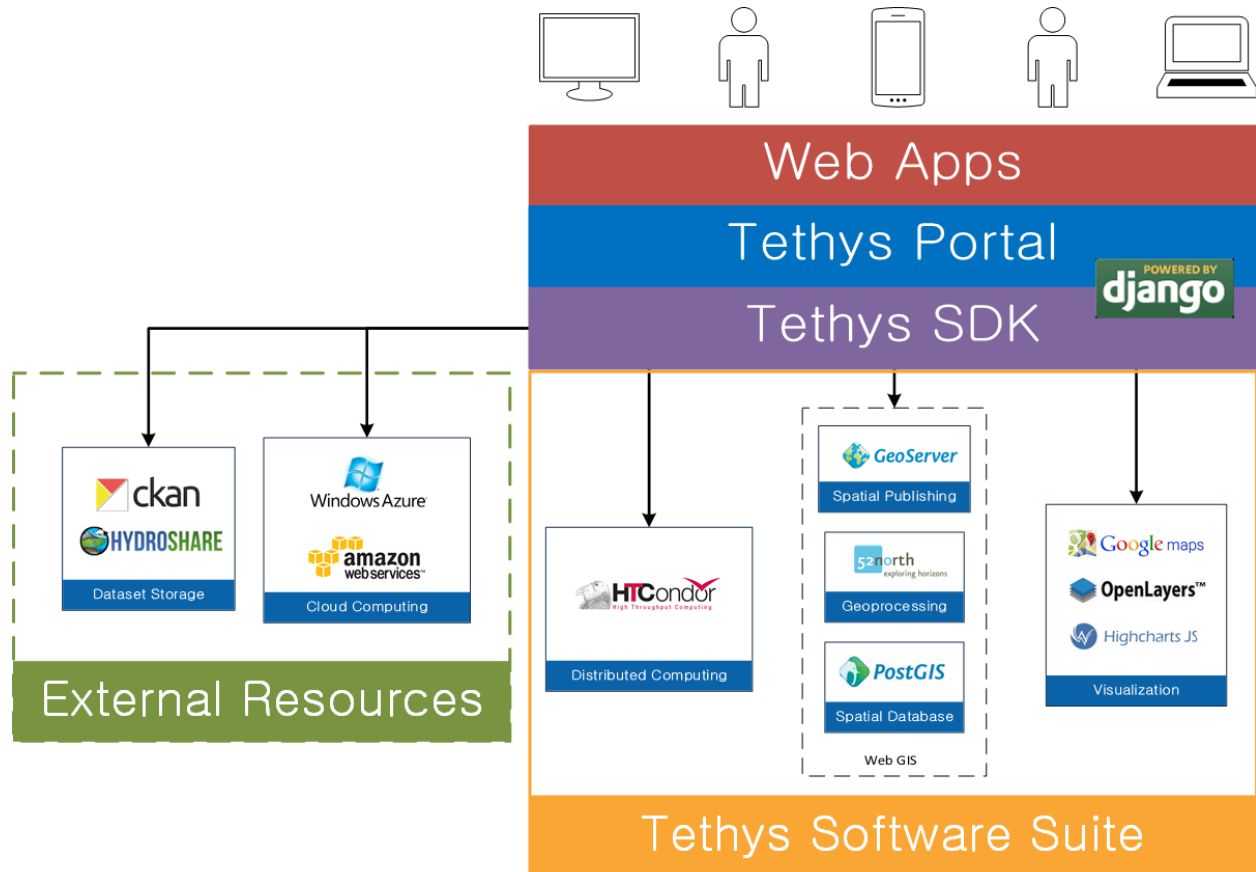
Stream Gage 1

Back

Tethys platform can be used to create engaging, interactive web apps for water resources.

1.1.1 Software Suite

Tethys Platform provides a suite of free and open source software. Included in the *Software Suite* is PostgreSQL with the PostGIS extension for spatial database storage, GeoServer for spatial data publishing, and 52 North WPS for geoprocessing. Tethys also provides Gizmos for inserting OpenLayers and Google Maps for interactive spatial data visualizations in your web apps. The *Software Suite* also includes HTCondor for managing distributed computing resources and scheduling computing jobs.



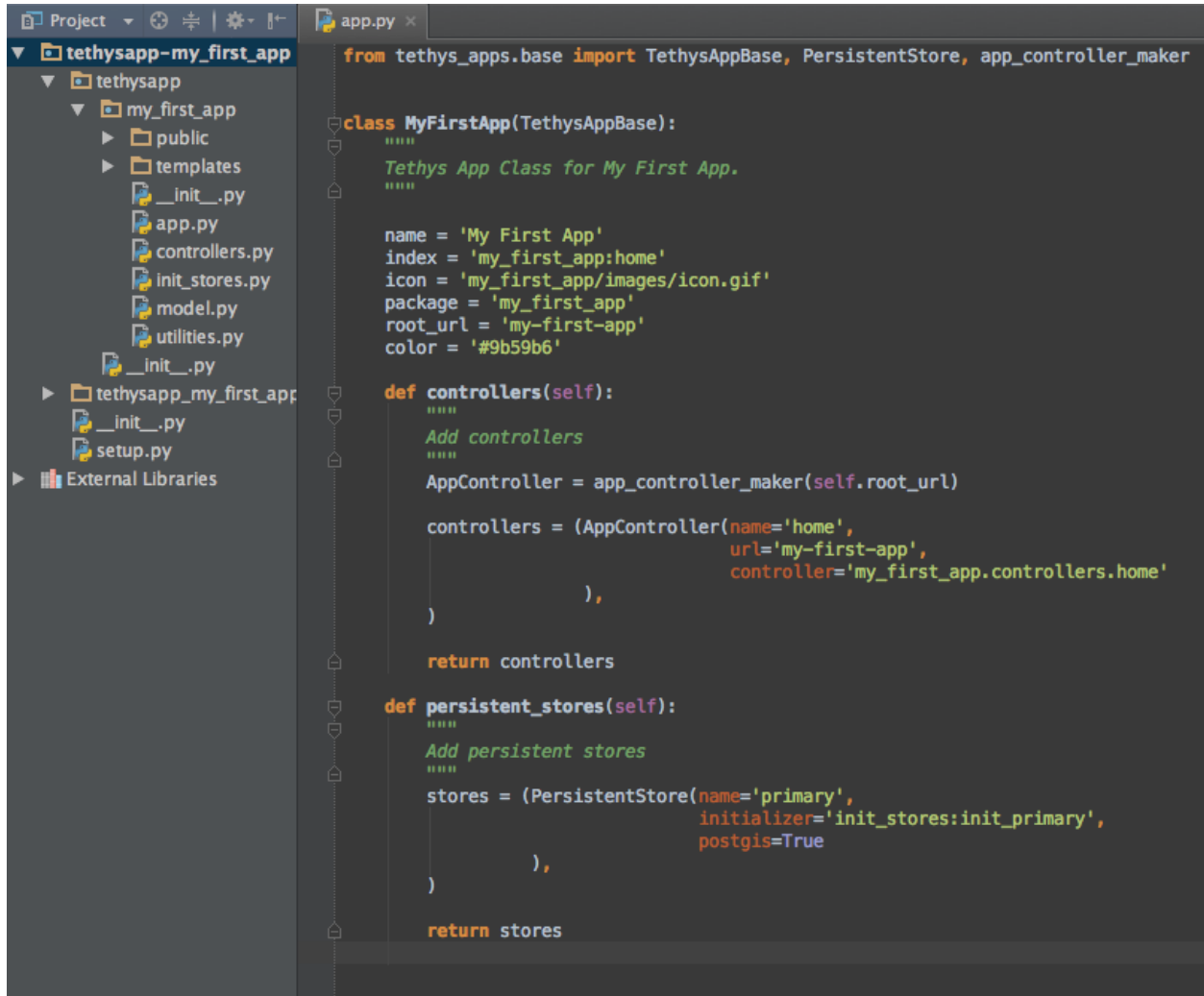
Tethys Platform include software to meet water resources web app development needs.

Note: Read more about the Software Suite by reading the *Software Suite* documentation.

1.1.2 Python Software Development Kit

Tethys web apps are developed with the Python programming language and a *Software Development Kit* (SDK). Tethys web apps projects are organized using a model-view-controller (MVC) approach. The SDK provides Python module links to each software component of the Tethys Platform, making the functionality of each software easy to incorporate each in your web apps. In addition, you can use all of the Python modules that you are accustomed to using in your scientific Python scripts to power your web apps.

Tethys web apps are developed using Python and the Tethys SDK.



The image shows a code editor interface. On the left, a file explorer displays the project structure for 'tethysapp-my_first_app'. The main editor area shows the code for 'app.py'.

```
from tethys_apps.base import TethysAppBase, PersistentStore, app_controller_maker

class MyFirstApp(TethysAppBase):
    """
    Tethys App Class for My First App.
    """

    name = 'My First App'
    index = 'my_first_app:home'
    icon = 'my_first_app/images/icon.gif'
    package = 'my_first_app'
    root_url = 'my-first-app'
    color = '#9b59b6'

    def controllers(self):
        """
        Add controllers
        """
        AppController = app_controller_maker(self.root_url)

        controllers = (AppController(name='home',
                                     url='my-first-app',
                                     controller='my_first_app.controllers.home'
                                     ),
                       )

        return controllers

    def persistent_stores(self):
        """
        Add persistent stores
        """
        stores = (PersistentStore(name='primary',
                                   initializer='init_stores:init_primary',
                                   postgis=True
                                   ),
                 )

        return stores
```

Note: Read more about the Tethys SDK by reading the *Software Development Kit* documentation.

1.1.3 Templating and Gizmos

Tethys SDK takes advantage of the Django template system so you can build dynamic pages for your web app while writing less HTML. It also provides a series of modular user interface elements called Gizmos. With only a few lines of code you can add range sliders, toggle switches, auto completes, interactive maps, and dynamic plots to your web app.

- [Quick Start](#)
- [Buttons](#)
- [Date Picker](#)
- [Range Slider](#)
- [Select Input](#)
- [Text Input](#)
- [Toggle Switch](#)
- [Message Box](#)
- [Table View](#)
- [Plot View](#)
- [Map View](#)**
- [Google Map View](#)
- [FetchClimate](#)

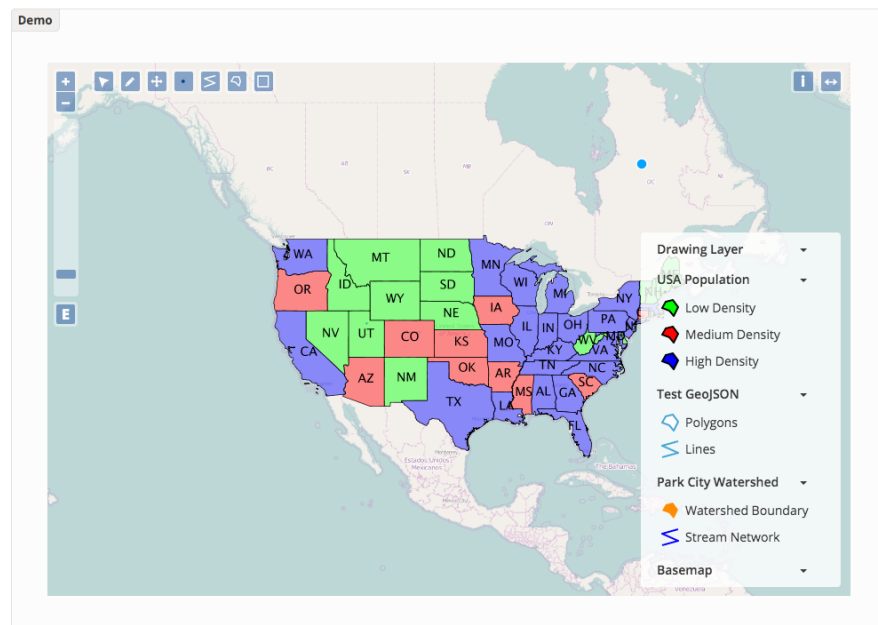
Map View

The Map View gizmo can be used to visualize maps of spatial data. Map View is powered by [OpenLayers 3](#), an open source pure javascript mapping library.

For example code and an explanation of options see Gizmo Options Object API for [Map View](#).

NOTE: Do not create more than one Map View gizmo on a page at any given time.

Click [here](#) for demo on a separate page.



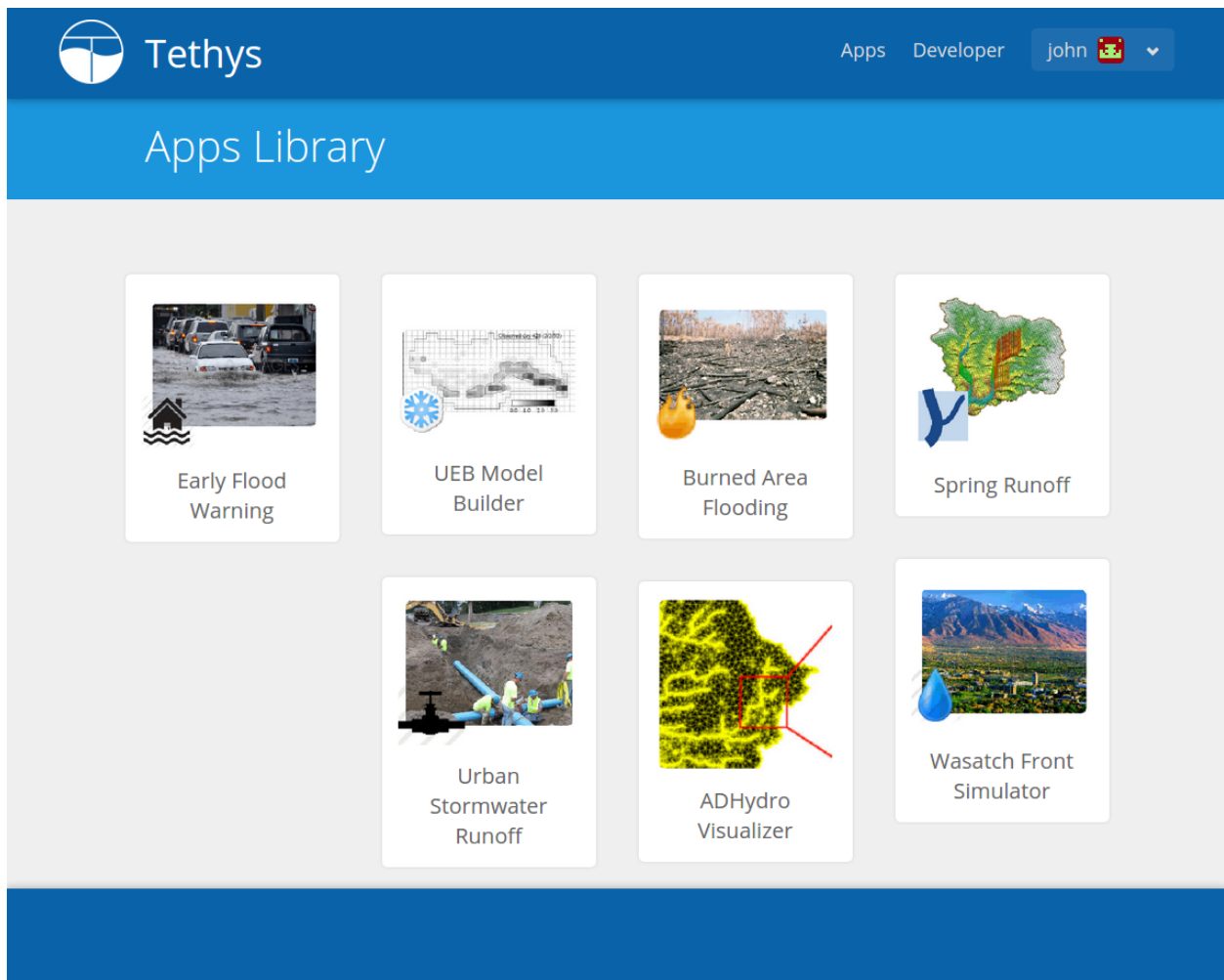
Insert common user interface elements like date pickers, maps, and plots with minimal coding.

Note: Read more about templating and Gizmo by reading the *App Templating API* and the *Template Gizmos API* documentation.

1.1.4 Tethys Portal

Tethys Platform includes a modern web portal built on Django that is used to host web apps called *Tethys Portal*. It provides the core website functionality that is often taken for granted in modern web applications including a user account system with a password reset mechanism for forgotten passwords. It provides an administrator backend that can be used to manage user accounts, permissions, link to elements of the software suite, and customize the instance.

The portal also includes landing page that can be used to showcase the capabilities of the Tethys Platform instance and an app library page that serves as the access point for installed apps. The homepage and theme of Tethys Portal are customizable allowing organizations to re-brand it to meet their needs.



Browse available web apps using the Apps Library.

Note: Read more about the Tethys Portal by reading the *Tethys Portal* documentation.

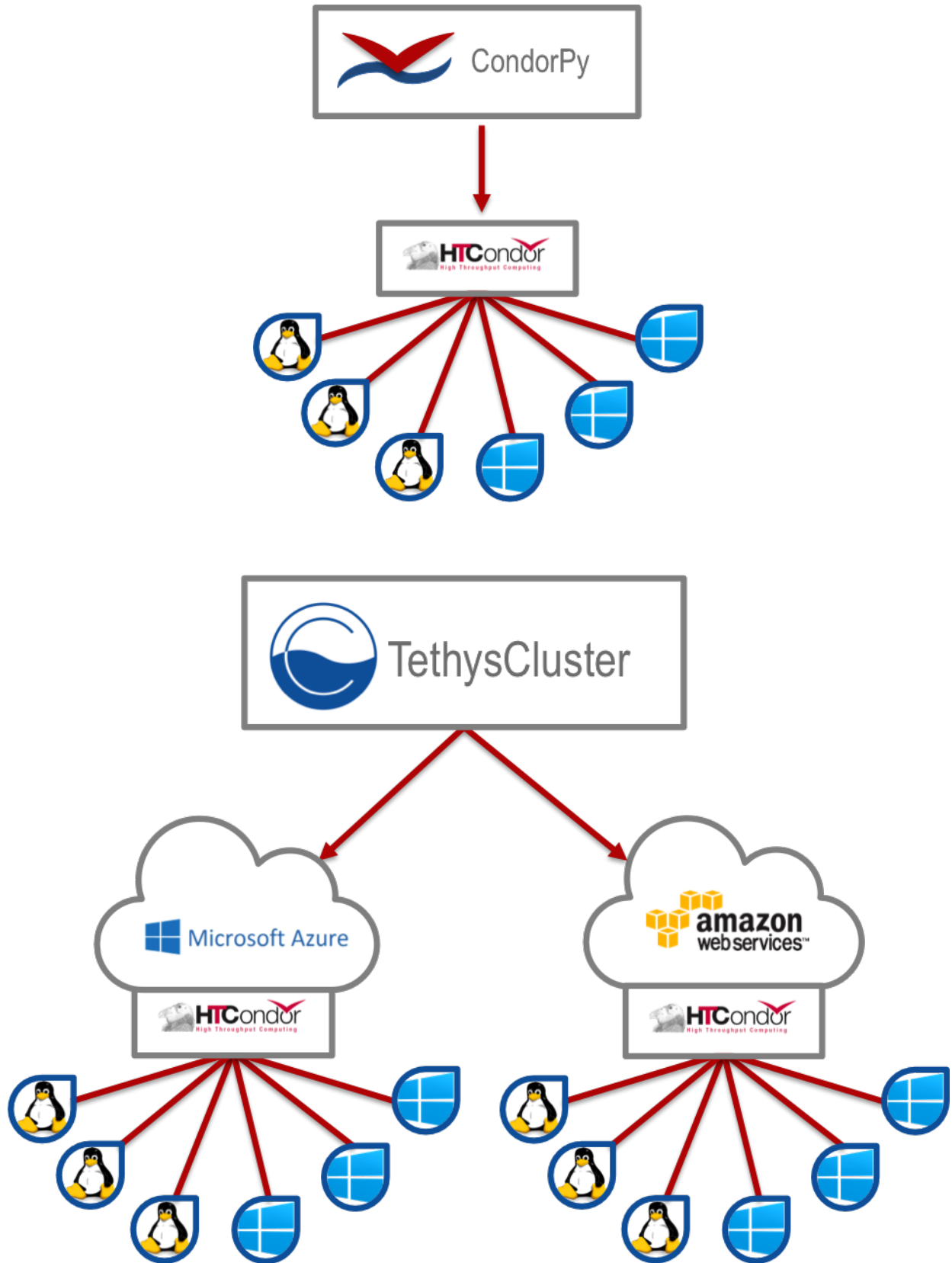
1.1.5 Computing

Tethys Platform includes Python modules that allow you to provision and run computing jobs in distributed computing environments. With CondorPy you can define your computing jobs and submit them to distributed computing environments provided by [HTCondor](#).

CondorPy enables computing jobs to be created and submitted to a HTCondor computing pool.

HTCondor provides a way to make use of the idle computing power that is already available in your office. Alternatively, TethysCluster enables you to provision scalable computing resources in the cloud using commercial services like [Amazon AWS](#) and [Microsoft Azure](#).

TethysCluster makes it easy to scale your computing resources using commercial cloud services.



Note: To learn more, read the *Distributed Computing API*.

1.1.6 Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1135482

1.2 What's New

Last Updated: May 28, 2015

Refer to this article for information about each new release of Tethys Platform.

1.2.1 Release 1.1.0

Gizmos

- Options objects for configuring gizmos (see *Template Gizmos API* and *Gizmo Options Object API* for more details).
- Many improvements to Map View (see *Map View*)
 - Improved layer support including GeoJSON, KML, WMS services, and ArcGIS REST services
 - Added a mechanism for creating legends
 - Added drawing capabilities
 - Upgraded to OpenLayers version 3.5.0
- New objects for simplifying Highcharts plot creation (see *Plot View*)
 - HighChartsLinePlot
 - HighChartsScatterPlot
 - HighChartsPolarPlot
 - HighChartsPiePlot
 - HighChartsBarPlot
 - HighChartsTimeSeries
 - HighChartsAreaRange
- Added the ability to draw a box on Google Map View

Tethys Portal Features

- Reset forgotten passwords
- Bypass the home page and redirect to apps library
- Rename the apps library page title
- The two mobile menus were combined into a single mobile menu

- Dataset Services and Web Processing Services admin settings combined into a single category called Tethys Services
- Added “Powered by Tethys Platform” attribution to footer

Job Manager

- Provides a unified interface for all apps to create submit and monitor computing jobs
- Abstracts the CondorPy module to provide a higher-level interface with computing jobs
- Allows definition of job templates in the app.py module of apps projects

Documentation Updates

- Added documentation about the Software Suite and the relationship between each software component and the APIs in the SDK is provided
- Documentation for manual download and installation of Docker images
- Added system requirements to documentation

Bug Fixes

- Naming new app projects during scaffolding is more robust
- Fixed bugs with fetch climate Gizmo
- Addressed issue caused by usernames that included periods (.) and other characters
- Made header more responsive to long names to prevent header from wrapping and obscuring controls
- Fixed bug with tethys gen apache command
- Addressed bug that occurred when naming WPS services with uppercase letters

Other

- Added parameter of UrlMap that can be used to specify custom regular expressions for URL search patterns
- Added validation to service engines
- Custom collectstatic command that automatically symbolically links the public/static directories of Tethys apps to the static directory
- Added “list” methods for dataset services and web processing services to allow app developers to list all available services registered on the Tethys Portal instance

1.3 Installation

Last Updated: April 18, 2015

This section describes how to install Tethys Platform. Installation instructions are provided for Linux (Ubuntu), Mac OSX, and Windows.

1.3.1 System Requirements

Last Updated: April 20, 2015

Use these guidelines as a starting point for installing Tethys Platform as a stand alone environment:

- Processor: 4 CPU Cores
- RAM: 4 GB
- Hard Disk: 10 GB

Caution: The stand alone configuration should be used primarily for development purposes. It is not recommended that you use a stand alone configuration for production installations. See the *Deploy* documentation for system requirements of a production installation.

1.3.2 Installation on Linux

Last Updated: May 22, 2015

Tip: To install and use Tethys Platform, you will need to be familiar with using the command line/terminal. For a quick introduction to the command line, see the *Terminal Quick Guide* article.

1. Install the Dependencies

1. If you are using a *Debian* based Linux operating system (like Ubuntu), you can install most of the dependencies via **apt-get**. Open a terminal and execute the following command:

```
$ sudo apt-get update
$ sudo apt-get install python-dev python-pip python-virtualenv libpq-dev libxml2-dev
libxslt1-dev
```

You may be prompted to enter your password to authorize the installation of these packages. If you are prompted about the disk space that will be used to install the dependencies, enter `Y` and press `Enter` to continue.

There will be a lot of text printed to the terminal as the dependencies are installed and it may take several minutes to complete. When it is finished you will see a normal terminal cursor again.

2. If you are not using a *Debian* based Linux operating system find the best way to install the following dependencies for your operating system:

Dependency	Description
Python	Python Programming Language, version 2.7.
pip	Python package management and installation tool.
virtualenv	virtualenv isolated Python environment creator.
git	Git open source distributed version control system.
docker	Docker virtual container system.
other libraries	libpq-dev, libxml2-dev, and libxslt1-dev

2. Finish the Docker Installation

There are a few additional steps that need to be completed to finish the installation of Docker.

1. Execute the following command to finish the installation of Docker:

```
$ source /etc/bash_completion.d/docker.io
```

2. Add your user to the Docker group. This is necessary to use the Tethys Docker commandline tools. In a command prompt execute:

```
$ sudo groupadd docker
$ sudo gpasswd -a ${USER} docker
$ sudo service docker.io restart
```

3. Close the terminal, then **log out** and **log back in** to make the changes take effect.

Important: **DO NOT FORGET PART C!** Be sure to logout of Ubuntu and log back in before you continue. You will not be able to complete the installation without completing this step.

Warning: Adding a user to the Docker group is the equivalent of declaring a user as root. See [Giving non-root access](#) for more details.

3. Install HTCondor (Optional)

HTCondor is a job scheduling and resource management system that is used by the Tethys Compute module. Distributed computing can be configured without installing HTCondor. For more information on how HTCondor is used for distributed computing in Tethys and the different configuration options see *Distributed Computing API*. Use one of the following links for instructions on how to install HTCondor through the package manager:

Enterprise Linux: [HTCondor YUM Repository](#)

Debian Linux: [HTCondor Debian Repository](#)

4. Create Virtual Environment and Install Tethys Platform

Python virtual environments are used to create isolated Python installations to avoid conflicts with dependencies of other Python applications on the same system. The following commands should be executed in a terminal.

1. Create a *Python virtual environment* and activate it:

```
$ sudo mkdir -p /usr/lib/tethys
$ sudo chown `whoami` /usr/lib/tethys
$ virtualenv --no-site-packages /usr/lib/tethys
$ . /usr/lib/tethys/bin/activate
```

Hint: You may be tempted to enter single quotes around the *whoami* directive above, but those characters are actually *grave accent* characters: ```. This key is usually located to the left of the `1` key or in that vicinity.

Important: The final command above activates the Python virtual environment for Tethys. You will know the virtual environment is active, because the name of it will appear in parenthesis in front of your terminal cursor:

```
(tethys) $ _
```

The Tethys virtual environment must remain active for the entire installation. If you need to logout or close the terminal in the middle of the installation, you will need to reactivate the virtual environment. This can be done at anytime by executing the following command (don't forget the dot):

```
$ . /usr/lib/tethys/bin/activate
```

2. Install Tethys Platform into the virtual environment with the following command:

```
$ git clone https://github.com/CI-WATER/tethys /usr/lib/tethys/src
```

Tip: If you would like to install a different version of Tethys Platform, you can use git to checkout the tagged release branch. For example, to checkout version 1.0.0:

```
cd /usr/lib/tethys/src
git checkout tags/1.0.0
```

For a list of all tagged releases, see [Tethys Platform Releases](#). Depending on the version you intend to install, you may need to delete your entire virtual environment (i.e.: the `/usr/lib/tethys` directory) to start fresh.

3. Install the Python modules that Tethys requires:

```
$ pip install --upgrade -r /usr/lib/tethys/src/requirements.txt
$ python /usr/lib/tethys/src/setup.py develop
```

4. Restart the Python virtual environment:

```
$ deactivate
$ . /usr/lib/tethys/bin/activate
```

5. Install Tethys Software Suite Using Docker

Tethys Platform provides a software suite that addresses the unique needs of water resources web app development including:

- PostgreSQL with PostGIS enabled for spatial database storage,
- 52 North WPS with GRASS and Sextante enabled for geoprocessing services, and
- GeoServer for spatial dataset publishing.

Installing some of these dependencies can be VERY difficult, so they have been provided as Docker containers to make installation EASY. The following instructions will walk you through installation of these software using Docker. See the [Docker Documentation](#) for more information about Docker containers.

Initialize the Docker Containers

Tethys provides set of commandline tools to help you manage the Docker containers. You must activate your Python environment to use the commandline tools. Execute the following Tethys commands using the **tethys** *Command Line Interface* to initialize the Docker containers:

```
$ tethys docker init
```

Tip: Running into errors with this command? Make sure you have completed all of step 2, including part c.

The first time you initialize the Docker containers, the images for each container will be downloaded. These images are large and it may take a long time for them to download.

After the images have been downloaded, the containers will automatically be installed. During installation, you will be prompted to enter various parameters needed to customize your instance of the software. Some of the parameters are usernames and passwords. **Take note of the usernames and passwords that you specify.** You will need them to complete the installation.

Start the Docker Containers

Use the following Tethys command to start the Docker containers:

```
$ tethys docker start
```

If you would like to test the Docker containers, see *Test Docker Containers*.

6. Create Settings File and Configure Settings

In the next steps you will configure your Tethys Platform and link it to each of the software in the software suite. Create a new settings file for your Tethys Platform installation using the **tethys** *Command Line Interface*. Execute the following command in the terminal:

```
$ tethys gen settings -d /usr/lib/tethys/src/tethys_apps
```

This will create a file called `settings.py` in the directory `/usr/lib/tethys/src/tethys_apps`. As the name suggests, the `settings.py` file contains all of the settings for the Tethys Platform. There are a few settings that need to be configured in this file.

Note: The `usr` directory is located in the root directory which can be accessed using a file browser and selecting Computer from the menu on the left.

Open the `settings.py` file that you just created (`/usr/lib/tethys/src/tethys_apps/settings.py`) in a text editor and modify the following settings appropriately.

1. Run the following command to obtain the host and port for Docker running the database (PostGIS). You will need these in the following steps:

```
$ tethys docker ip
```

2. Replace the password for the main Tethys Portal database, **tethys_default**, with the password you created in the previous step. Also make sure that the host and port match those given from the `tethys docker ip` command (PostGIS). This is done by changing the values of the `PASSWORD`, `HOST`, and `PORT` parameters of the `DATABASES` setting:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'tethys_default',
        'USER': 'tethys_default',
        'PASSWORD': 'pass',
        'HOST': '127.0.0.1',
        'PORT': '5435'
    }
}
```

3. Find the `TETHYS_DATABASES` setting near the bottom of the file and set the `PASSWORD` parameters with the passwords that you created in the previous step. If necessary, also change the `HOST` and `PORT` to match the host and port given by the `tethys docker ip` command for the database (PostGIS):

```
TETHYS_DATABASES = {
    'tethys_db_manager': {
        'NAME': 'tethys_db_manager',
        'USER': 'tethys_db_manager',
        'PASSWORD': 'pass',
        'HOST': '127.0.0.1',
```

```
        'PORT': '5435'
    },
    'tethys_super': {
        'NAME': 'tethys_super',
        'USER': 'tethys_super',
        'PASSWORD': 'pass',
        'HOST': '127.0.0.1',
        'PORT': '5435'
    }
}
```

4. Save your changes and close the `settings.py` file.

7. Create Database Tables

Execute the **tethys manage syncdb** command from the Tethys *Command Line Interface* to create the database tables. In the terminal:

```
$ tethys manage syncdb
```

Important: When prompted to create a system administrator enter 'yes'. Take note of the username and password, as this will be the user you use to manage your Tethys Platform installation.

8. Start up the Django Development Server

You are now ready to start the development server and view your instance of Tethys Platform. The website that comes with Tethys Platform is called Tethys Portal. In the terminal, execute the following command from the Tethys *Command Line Interface*:

```
$ tethys manage start
```

Open <http://localhost:8000/> in a new tab in your web browser and you should see the default Tethys Portal landing page.

9. Web Admin Setup

You are now ready to configure your Tethys Platform installation using the web admin interface. Follow the *Web Admin Setup* tutorial to finish setting up your Tethys Platform.

1.3.3 Installation on Mac OSX

Last Updated: May 27, 2015

Use these instructions to install a development environment on OSX. These instructions have been tested with OSX Yosemite.

Tip: To install and use Tethys Platform, you will need to be familiar with using the command line/terminal. For a quick introduction to the command line, see the *Terminal Quick Guide* article.



Welcome to Tethys Portal,
the hub for your apps.

Sign Up

Tethys Portal is designed to be customizable, so that you can host apps for your organization. You can change everything on this page from the Home Page settings.

1. Install the Dependencies

1. Many of the commandline tools for the Mac are provided through Xcode. You will need to install or update the Xcode commandline tools by opening a Terminal and executing the following command:

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

Follow the prompts to download and install commandline developer tools.

Note: If you do not have Xcode installed, you may need to install it before running this command. You can install it using the Mac App Store.

2. One feature that Mac OSX lacks that many Linux distributions provide is a package manager on the commandline. [Homebrew](#) is an excellent package manager for Mac OSX and you will use it to install the Tethys dependencies. Install [Homebrew](#) by executing the following command in a Terminal:

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

After [Homebrew](#) is installed, you will need to add a few entries to the PATH variable. In a terminal, execute the following command:

```
$ sudo nano /etc/paths
```

- Enter your password if prompted.
- Add the following two paths at the **top of the file** if they are not already present

```
/usr/local/bin  
/usr/local/sbin
```

- Press `control-x` to quit and enter `Y` to save the changes.

Note: You may need to close Terminal and open it again for these changes to take effect.

3. You will need a fresh installation of Python with the `pip` and `virtualenv` installed. Fortunately, [Homebrew](#) automatically installs `pip` and `virtualenv` with its Python package. Install Python and other dependencies using [Homebrew](#) as follows:

```
$ brew install git libpqxx libxml2 libxslt libffi
$ brew unlink openssl
$ brew install https://raw.githubusercontent.com/Homebrew/homebrew/
62fc2a1a65e83ba9dbb30b2e0a2b7355831c714b/Library/Formula/openssl.rb$ brew link
--force openssl && brew switch openssl 1.0.1j_1
$ brew install python --with-brewed-openssl --build-from-source
$ pip install virtualenv
```

Tip: If you encounter trouble using [Homebrew](#) to install these dependencies, run the following command in the Terminal:

```
$ brew doctor
```

This will generate a list of suggestions for remedying your [Homebrew](#) installation.

2. Create Virtual Environment and Install Tethys Platform

Python virtual environments are used to create isolated Python installations to avoid conflicts with dependencies of other Python applications on the same system. Execute the following commands in Terminal.

1. Create a *Python virtual environment* and activate it:

```
$ sudo mkdir -p /usr/lib/tethys
$ sudo chown `whoami` /usr/lib/tethys
$ virtualenv --no-site-packages /usr/lib/tethys
$ . /usr/lib/tethys/bin/activate
```

Important: The final command above activates the Python virtual environment for Tethys. You will know the virtual environment is active, because the name of it will appear in parenthesis in front of your terminal cursor:

```
(tethys) $ _
```

The Tethys virtual environment must remain active for most of the installation. If you need to logout or close the terminal in the middle of the installation, you will need to reactivate the virtual environment. This can be done at anytime by executing the following command (don't forget the dot):

```
$ . /usr/lib/tethys/bin/activate
```

As a reminder, the commands requiring your Tethys virtual environment be active will show the cursor with “(tethys)” next to it.

2. Install Tethys Platform into the virtual environment with the following command:

```
(tethys) $ git clone https://github.com/CI-WATER/tethys /usr/lib/tethys/src
```

Tip: If you would like to install a different version of Tethys Platform, you can use `git` to checkout the tagged release branch. For example, to checkout version 1.0.0:

```
cd /usr/lib/tethys/src
git checkout tags/1.0.0
```

For a list of all tagged releases, see [Tethys Platform Releases](#). Depending on the version you intend to install, you may need to delete your entire virtual environment (i.e.: the `/usr/lib/tethys` directory) to start fresh.

3. Install the Python modules that Tethys requires:

```
(tethys) $ pip install --upgrade -r /usr/lib/tethys/src/requirements.txt
(tethys) $ python /usr/lib/tethys/src/setup.py develop
```

4. Restart the Python virtual environment:

```
(tethys) $ deactivate
          $ . /usr/lib/tethys/bin/activate
```

3. Install Tethys Software Suite Using Docker

Tethys Platform provides a software suite that addresses the unique needs of water resources web app development (see [Features](#) for more details). To make installation of the software easy, each software has been provided as Docker container. The following instructions will walk you through installation of these software using Docker. See the [Docker Documentation](#) for more information about Docker.

1. Install Boot2Docker version 1.6 using the [Install Docker on Mac OSX instructions](#). Look for the heading titled *Install Boot2Docker*. Verify the installation using the instructions using the instructions under the *Start the Boot2Docker Application* heading.
2. Close the Boot2Docker terminal and open a new one. Initialize the Tethys Docker containers with the following command:

```
          $ . /usr/lib/tethys/bin/activate
(tethys) $ tethys docker init
```

Follow the interactive prompts to customize your Docker installations. To accept the default value shown in square brackets, simply press `enter`. **Take note of any passwords you are prompted to create.**

Note: The first time you initialize the Docker containers, the images for each container will need to be downloaded. These images are large and it may take a long time for them to download.

3. Start the docker containers with the following command:

```
(tethys) $ tethys docker start
```

After running the `tethys docker start` command, you will have the following software running:

- PostgreSQL with PostGIS
- 52 North WPS
- GeoServer

If you would like to test the Docker containers, see the [Test Docker Containers](#) article.

Note: Although each Docker container appears to start instantaneously, it may take several minutes for the started containers to be fully up and running.

4. Install HTCondor (Optional)

HTCondor is a job scheduling and resource management system that is used by the Tethys Compute module. Distributed computing can be configured without installing HTCondor. For more information on how HTCondor is used for distributed computing in Tethys and the different configuration options see *Distributed Computing API*.

1. Use a browser to download the HTCondor tarball from the [HTCondor downloads page](#). Click the link next to the version you wish to install. Select `condor-X.X.X-x86_64_MacOSX-stripped.tar.gz`, complete the rest of the form to submit your download request. This should redirect you to a page with a link to download the tarball.
2. In a terminal change directories to the location of the tarball, untar it, and change into the new directory:

```
$ tar xzf condor-X.X.X-x86_64_MacOSX-stripped.tar.gz
$ cd condor-X.X.X-x86_64_MacOSX7-stripped
```

3. Run the perl script `condor_install` with the following options to install condor

```
$ perl condor_install --install --install-dir /usr/local/condor
```

4. Add an environmental variable to point to the location of the global `condor_config` file, and add the `condor bin` and `sbin` directories to `PATH`. This can be done by executing the `condor.sh` script that was generated when condor was installed:

```
. /usr/local/condor/condor.sh
```

Tip: To have these environmental variables exported automatically when a terminal is started add the previous command to the `.bash_profile`.

```
echo '. /usr/local/condor/condor.sh' >> ~/.bash_profile
```

5. Start condor:

```
$ condor_master
```

6. Check that condor is running:

```
$condor_status
```

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
slot1@ciwater-1.1.0	OSX	X86_64	Unclaimed	Idle	0.000	1024	0+00:50:05
slot2@ciwater-1.1.0	OSX	X86_64	Unclaimed	Idle	0.660	1024	0+00:50:06
slot3@ciwater-1.1.0	OSX	X86_64	Unclaimed	Idle	0.000	1024	0+00:50:07
slot4@ciwater-1.1.0	OSX	X86_64	Unclaimed	Idle	0.000	1024	0+00:50:08
slot5@ciwater-1.1.0	OSX	X86_64	Unclaimed	Idle	0.000	1024	0+00:50:09
slot6@ciwater-1.1.0	OSX	X86_64	Unclaimed	Idle	0.000	1024	0+00:50:10
slot7@ciwater-1.1.0	OSX	X86_64	Unclaimed	Idle	0.000	1024	0+00:50:11
slot8@ciwater-1.1.0	OSX	X86_64	Unclaimed	Idle	1.000	1024	0+00:50:04
Total Owner Claimed Unclaimed Matched Preempting Backfill							
	X86_64/OSX	8	0	0	8	0	0
	Total	8	0	0	8	0	0

5. Create Settings File and Configure Settings

Create a settings file for your Tethys Platform installation using the **tethys** *Command Line Interface*. Execute the following command in the terminal:

```
(tethys) $ tethys gen settings -d /usr/lib/tethys/src/tethys_apps
```

This will create a file called `settings.py` in the directory `/usr/lib/tethys/src/tethys_apps`. Open the `settings.py` file and make the following modifications.

Note: Accessing the `settings.py` file can be done by opening a new Finder Window and selecting `Go > Go to Folder...` from the menu. Enter `/usr/lib/tethys/src/tethys_apps` in the text box and press `Go` to browse to directory. From here you can open the `settings.py` file using your favorite text editor.

1. Run the following command to obtain the host and port for the Docker running the database:

```
(tethys) $ tethys docker ip
```

```
PostGIS/Database:
  Host: 192.168.59.103
  Port: 5435
...
```

2. Open the `settings.py` and locate the `DATABASES` setting. Replace the password for `tethys_default`, with the password you created when initializing the Docker containers. Also set the host and port to match those given from the `tethys docker ip` command:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycpg2',
        'NAME': 'tethys_default',
        'USER': 'tethys_default',
        'PASSWORD': 'pass',
        'HOST': '192.168.59.103',
        'PORT': '5435'
    }
}
```

3. Find the `TETHYS_DATABASES` setting near the bottom of the `settings.py` file and set the passwords for the `tethys_db_manager` and `tethys_super` database users. If necessary, also change the `HOST` and `PORT` to match the host and port given by the `tethys docker ip` command:

```
TETHYS_DATABASES = {
    'tethys_db_manager': {
        'NAME': 'tethys_db_manager',
        'USER': 'tethys_db_manager',
        'PASSWORD': 'pass',
        'HOST': '192.168.59.103',
        'PORT': '5435'
    },
    'tethys_super': {
        'NAME': 'tethys_super',
        'USER': 'tethys_super',
        'PASSWORD': 'pass',
        'HOST': '192.168.59.103',
        'PORT': '5435'
    }
}
```

4. Save your changes and close the `settings.py` file.

6. Create Database Tables

Execute the **tethys manage syncdb** command from the Tethys *Command Line Interface* to create the database tables. In the terminal:

```
(tethys) $ tethys manage syncdb
```

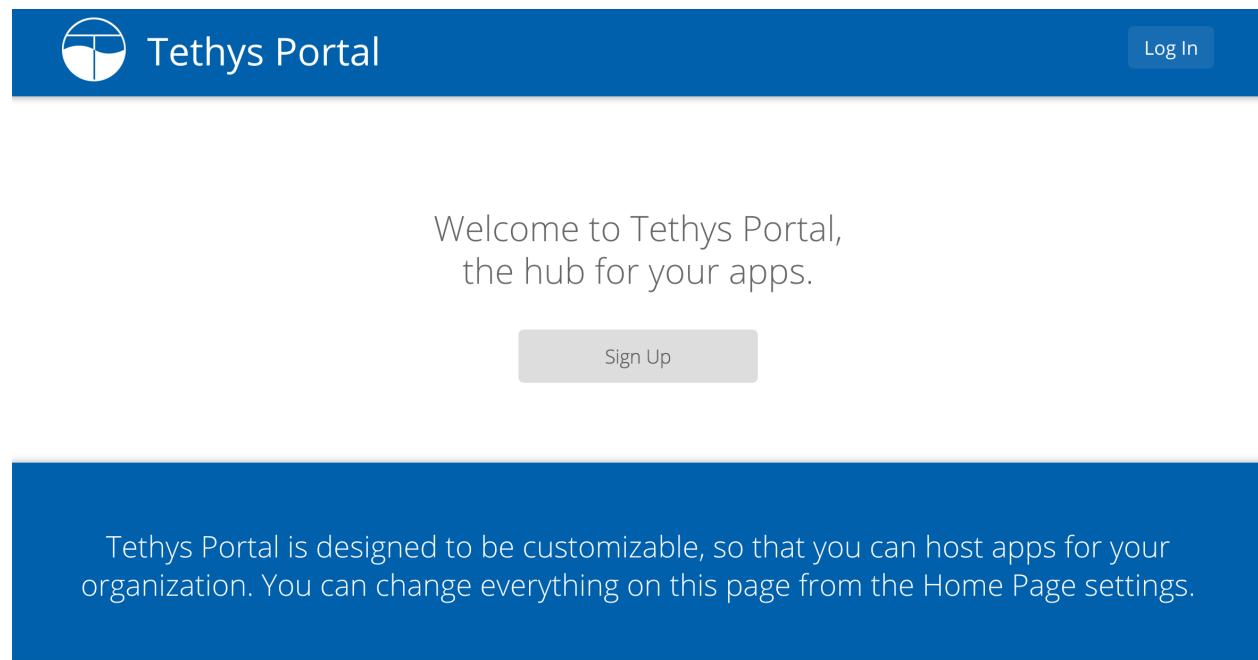
Important: When prompted to create a system administrator enter 'yes'. Take note of the username and password, as this will be the administrator user you will use to manage your Tethys Platform installation.

7. Start up the Django Development Server

You are now ready to start the development server and view your instance of Tethys Platform. In the terminal, execute the following command from the Tethys *Command Line Interface*:

```
(tethys) $ tethys manage start
```

Tethys Platform provides a web interface that is called the Tethys Portal. You can access your Tethys Portal by opening <http://localhost:8000/> in a new tab in your web browser.



8. Web Admin Setup

You are now ready to configure your Tethys Platform installation using the web admin interface. Follow the *Web Admin Setup* tutorial to finish setting up your Tethys Platform.

1.3.4 Web Admin Setup

Last Updated: February 2, 2015

The final step required to setup your Tethys Platform is to link it to the software that is running in the Docker containers. This is done using the Tethys Portal Admin console.

1. Access Tethys Portal Admin Console

The Tethys Portal Admin Console is only accessible to users with administrator rights. When you initialized the database, you were prompted to create the default admin user. Use these credentials to log in for the first time.

1. Use the “Log In” link on the Tethys Portal homepage to log in as an administrator. Use the username and password that you setup when you initialized the database.

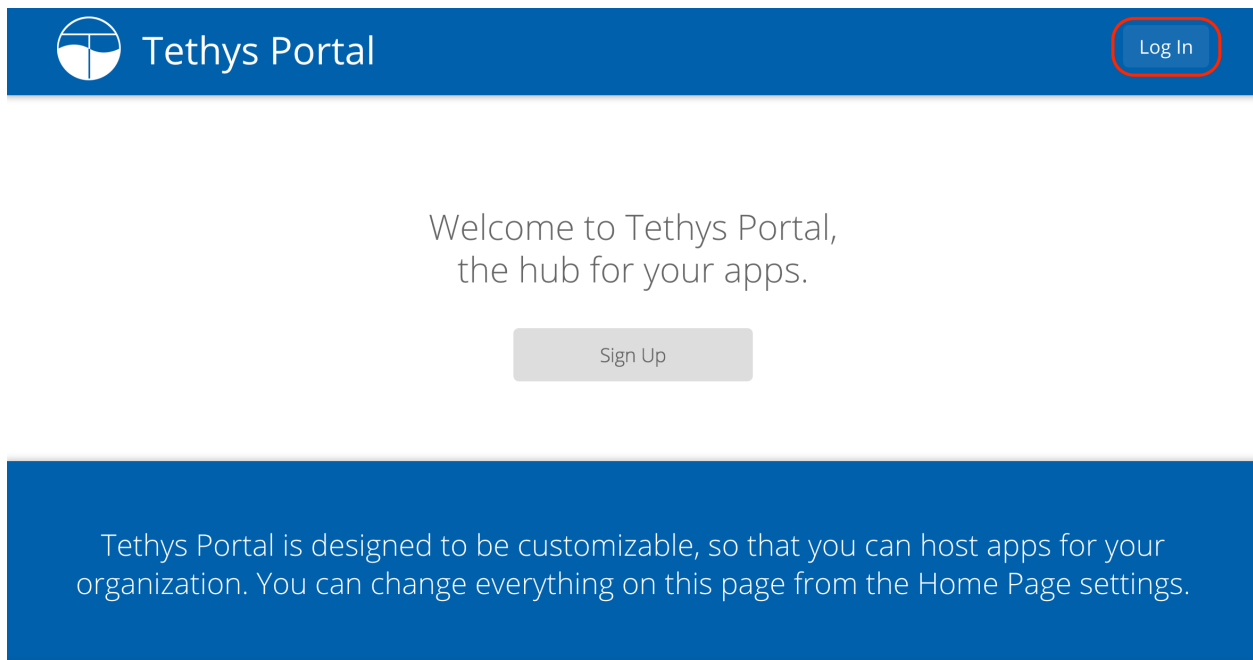


Figure 1.1: **Figure 1.** Log In link for Tethys Portal.

2. Select “Site Admin” from the user drop down menu (Figure 1).

You will now be looking at the Tethys Portal Web Admin Console. The Web Admin console can be used to manage user accounts, customize the homepage of your Tethys Portal, and configure the software included in Tethys Platform. Take a moment to familiarize yourself with the different options that are available in the Web Admin (Figure 2).

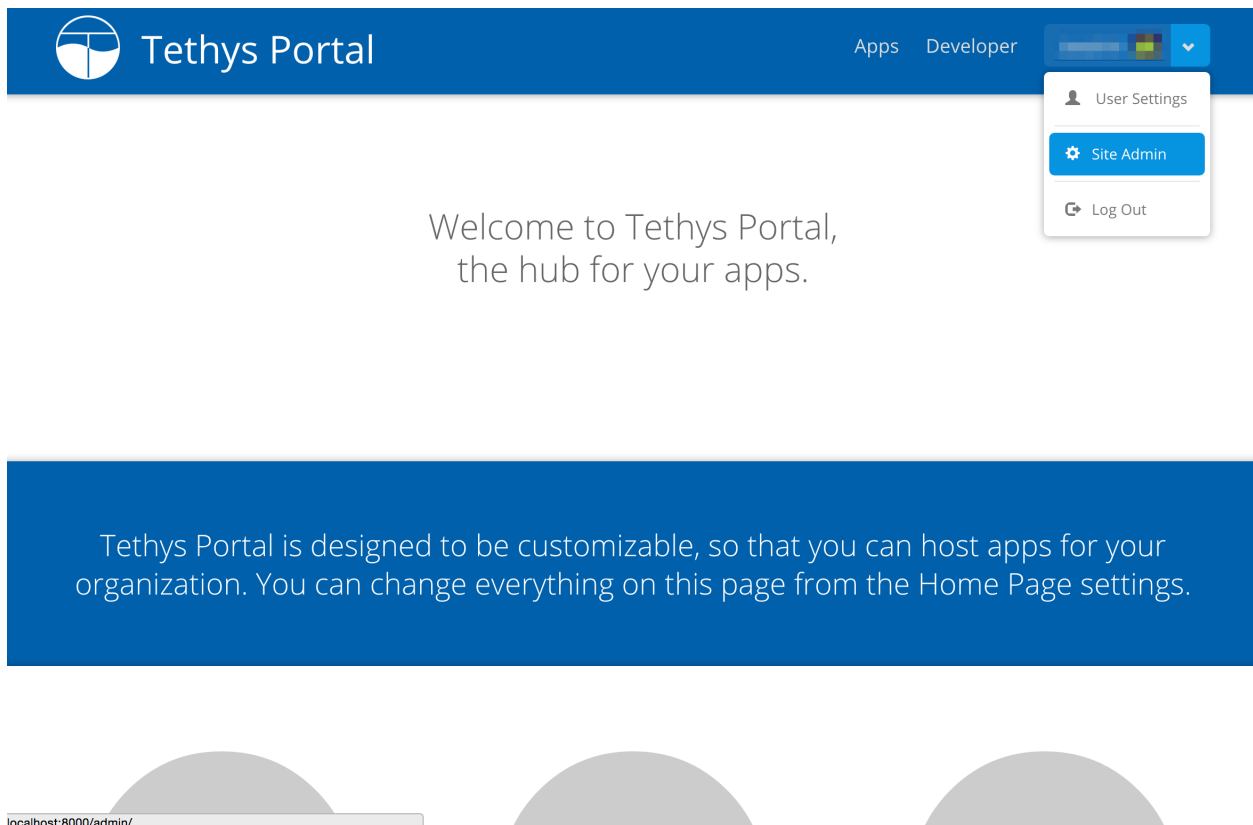
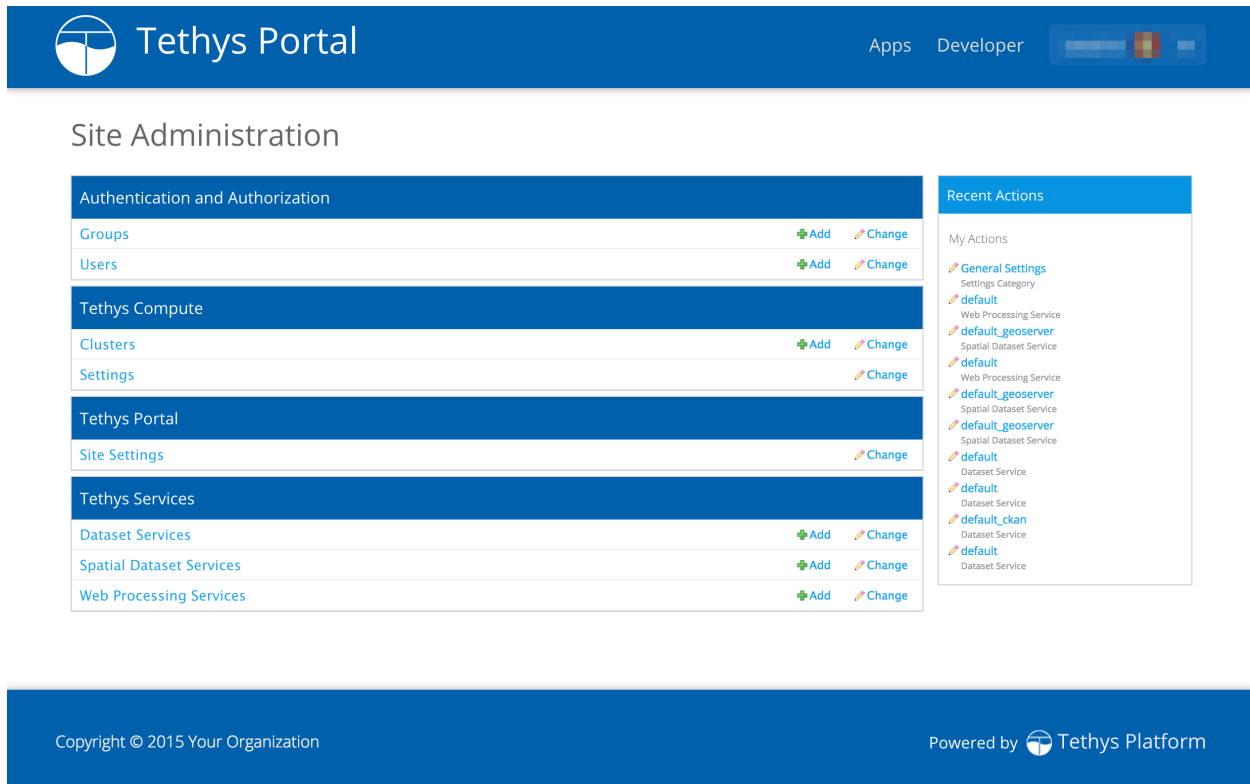


Figure 1.2: **Figure 2.** Select “Site Admin” from the user dropdown menu.

Figure 1.3: **Figure 3.** The Tethys Portal Web Admin Console.

2. Link to 52 North WPS Docker

The built in 52 North Web Processing Service (WPS) is provided as one mechanism for Geoprocessing in apps. It exposes the GRASS GIS and Sextante geoprocessing libraries as web services. See *Web Processing Services API* documentation for more details about how to use 52 North WPS processing in apps. Complete the following steps to link Tethys with the 52 North WPS:

1. Select “Web Processing Services” from the options listed on the Tethys Portal Admin Console.
2. Click on the “Add Web Processing Service” button to create a new link to the web processing service.
3. Provide a unique name for the web processing service.
4. Provide an endpoint to the 52 North WPS that is running in Docker. The endpoint is a URL pointing to the WPS API. The endpoint will be of the form: `http://<host>:<port>/wps/WebProcessingService`.

Execute the following command in the terminal to determine the endpoint for the built-in GeoServer:

```
(tethys)$ tethys docker ip
...
52 North WPS:
  Host: 192.168.59.103
  Port: 8282
  Endpoint: http://192.168.59.103:8282/wps/WebProcessingService
```

When you are done you will have something similar to this:

5. Press “Save” to save the WPS configuration.

The screenshot shows the Tethys Portal interface. At the top, there is a blue header with the Tethys Portal logo and navigation links for 'Apps' and 'Developer'. Below the header, the breadcrumb trail reads 'Home > Tethys Services > Web Processing Services'. The main heading is 'Select Web Processing Service To Change'. On the right side of this heading, there is a blue button labeled 'Add Web Processing Service +'. Below the heading is a table with the following content:

Action:	Go	0 of 2 selected
<input type="checkbox"/>	Web Processing Service	
<input type="checkbox"/>	default_wps	
<input type="checkbox"/>	default	

At the bottom of the table, it says '2 Web Processing Services'. The footer of the page contains 'Copyright © 2015 Your Organization' and 'Powered by Tethys Platform'.

Figure 1.4: **Figure 4.** Select the “Add Web Processing Service” button.

The screenshot shows the Tethys Portal interface. At the top, there is a blue header with the Tethys Portal logo and navigation links for 'Apps' and 'Developer'. Below the header, the breadcrumb trail reads 'Home > Tethys Services > Web Processing Services > default_wps'. The main heading is 'Change Web Processing Service'. On the right side of this heading, there is a blue button labeled 'History'. Below the heading is a form with the following fields:

Name: default_wps

Endpoint: http://192.168.59.103:8282/wps/WebProc

Username: wps

Password: ...

At the bottom left of the form, there is a red button labeled 'Delete'. At the bottom right, there are three buttons: 'Save and add another', 'Save and continue editing', and a green 'Save' button.

The footer of the page contains 'Copyright © 2015 Your Organization' and 'Powered by Tethys Platform'.

Figure 1.5: **Figure 5.** Fill out the form to register a new Web Processing Service.

3. Link to GeoServer

Tethys Platform provides GeoServer as a built-in Spatial Dataset Service. Spatial Dataset Services can be used by apps to publish Shapefiles and other spatial files as web resources. See *Spatial Dataset Services API* documentation for how to use Spatial Dataset Services in apps. To link your Tethys Platform to the built-in GeoServer or an external Spatial Dataset Service, complete the following steps.

1. Select “Spatial Dataset Services” from the options listed on the Tethys Portal Admin Console.
2. Click on the “Add Spatial Dataset Service” button to create a new spatial dataset service.

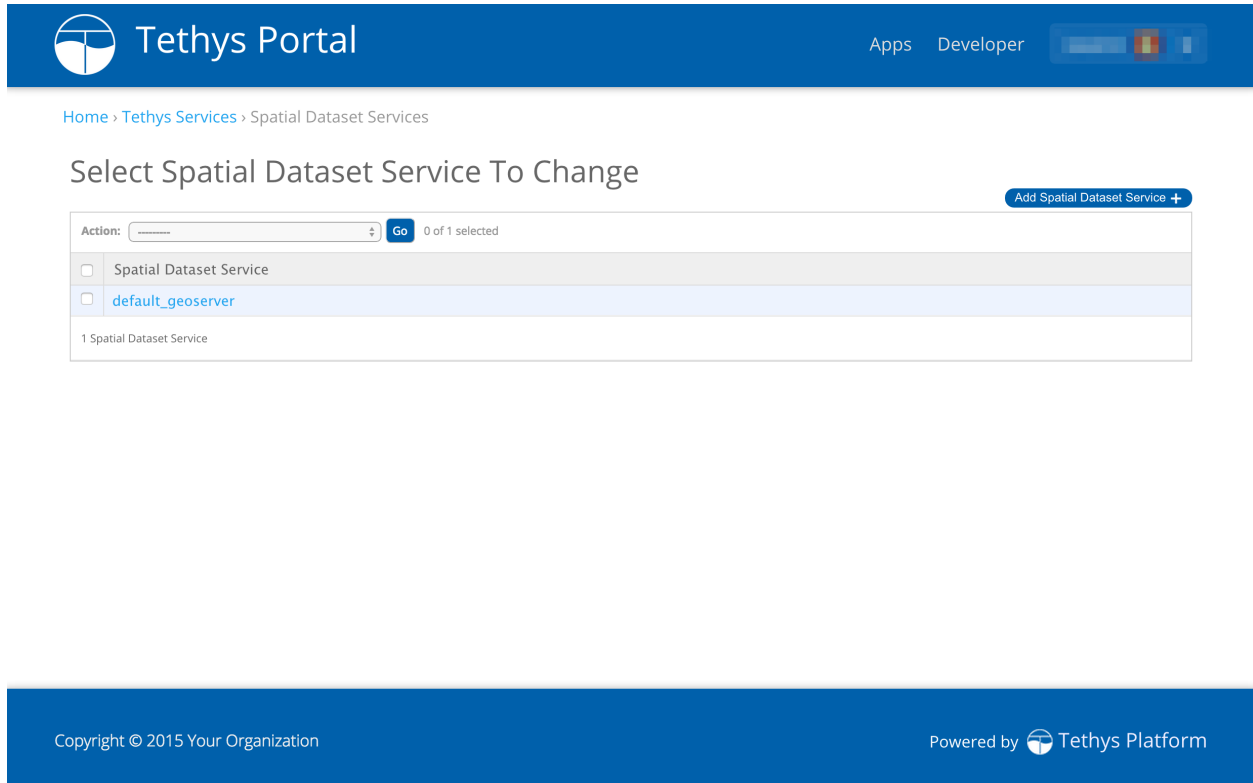


Figure 1.6: **Figure 4.** Select the “Add Spatial Dataset Service” button.

3. Provide a unique name for the spatial dataset service.
4. Select “GeoServer” as the engine and provide an endpoint to the Spatial Dataset Service. The endpoint is a URL pointing to the API of the Spatial Dataset Service. For GeoServers, this endpoint is of the form: `http://<host>:<port>/geoserver/rest`.

Execute the following command in the terminal to determine the endpoint for the built-in GeoServer:

```
(tethys)$ tethys docker ip
...
GeoServer:
  Host: 127.0.0.1
  Port: 8181
  Endpoint: http://127.0.0.1:8181/geoserver/rest
...
```

5. Specify either the username or password of your GeoServer as well. The default GeoServer username and

password are “*admin*” and “*geoserver*”, respectively. When you are done you will have something similar to this:

The screenshot shows the Tethys Portal interface for editing a Spatial Dataset Service. The form is titled "Change Spatial Dataset Service" and includes a "History" button. The form fields are as follows:

- Name:** default_geoserver
- Engine:** GeoServer
- Endpoint:** http://192.168.59.103:8181/geoserver/res
- Apikey:** (empty)
- Username:** admin
- Password:** (masked with dots)

At the bottom of the form, there are four buttons: "Delete" (with a red asterisk icon), "Save and add another", "Save and continue editing", and "Save" (in a green box).

Figure 1.7: **Figure 5.** Fill out the form to register a new Spatial Dataset Service.

6. Press “Save” to save the Spatial Dataset Service configuration.

4. Link to Dataset Services

Optionally, you may wish to link to external Dataset Services such as CKAN and HydroShare. Dataset Services can be used by apps as data stores and data sources. See [Dataset Services API](#) documentation for how to use Dataset Services in apps. Complete the following steps for each dataset service you wish to link to:

1. Select “Dataset Services” from the options listed on the Tethys Portal Admin Console.
2. Click on the “Add Dataset Service” button to create a new link to the dataset service.
3. Provide a unique name for the dataset service.
4. Select the appropriate engine and provide an endpoint to the Dataset Service. The endpoint is a URL pointing to the dataset service API. For example, the endpoint for a CKAN dataset service would be of the form `http://<host>:<port>/api/3/action`.

If authentication is required, specify either the API Key or username or password as well. When you are done you will have something similar to this:

Tip: When linking Tethys to a CKAN dataset service, an API Key is required. All user accounts are

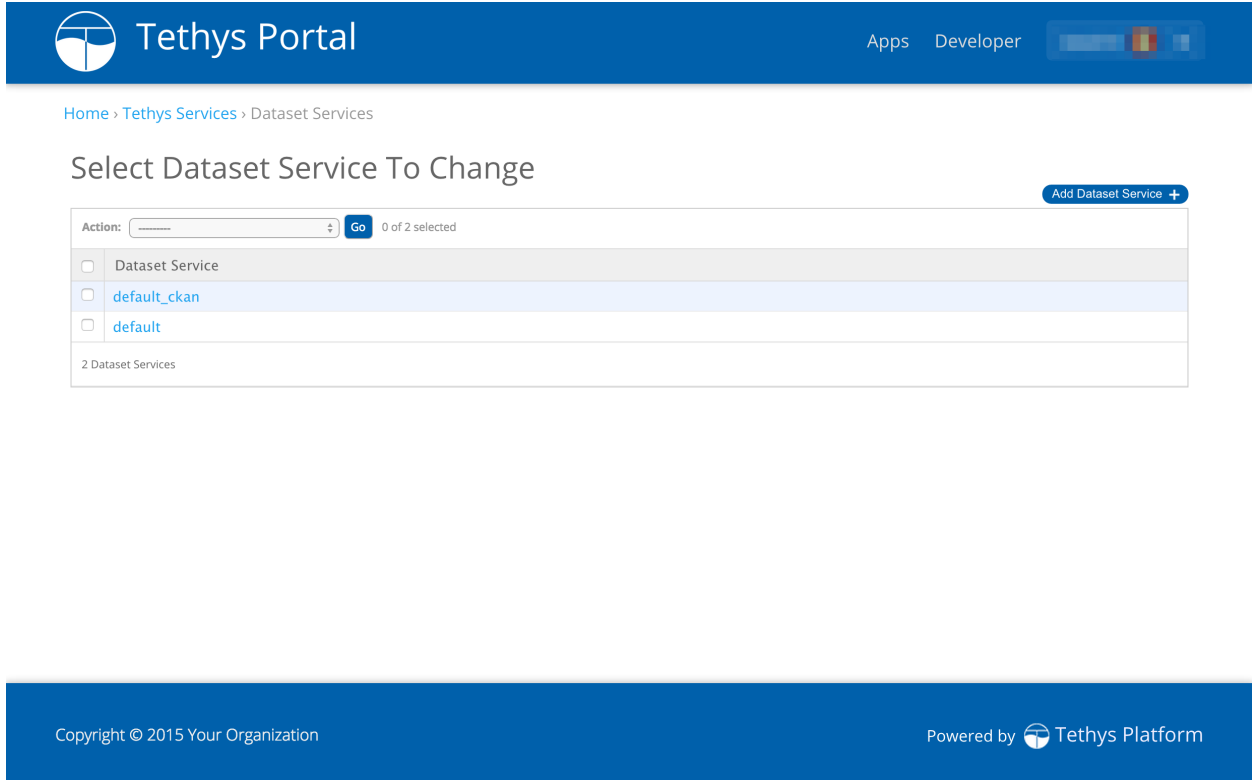


Figure 1.8: **Figure 4.** Select the “Add Dataset Service” button.

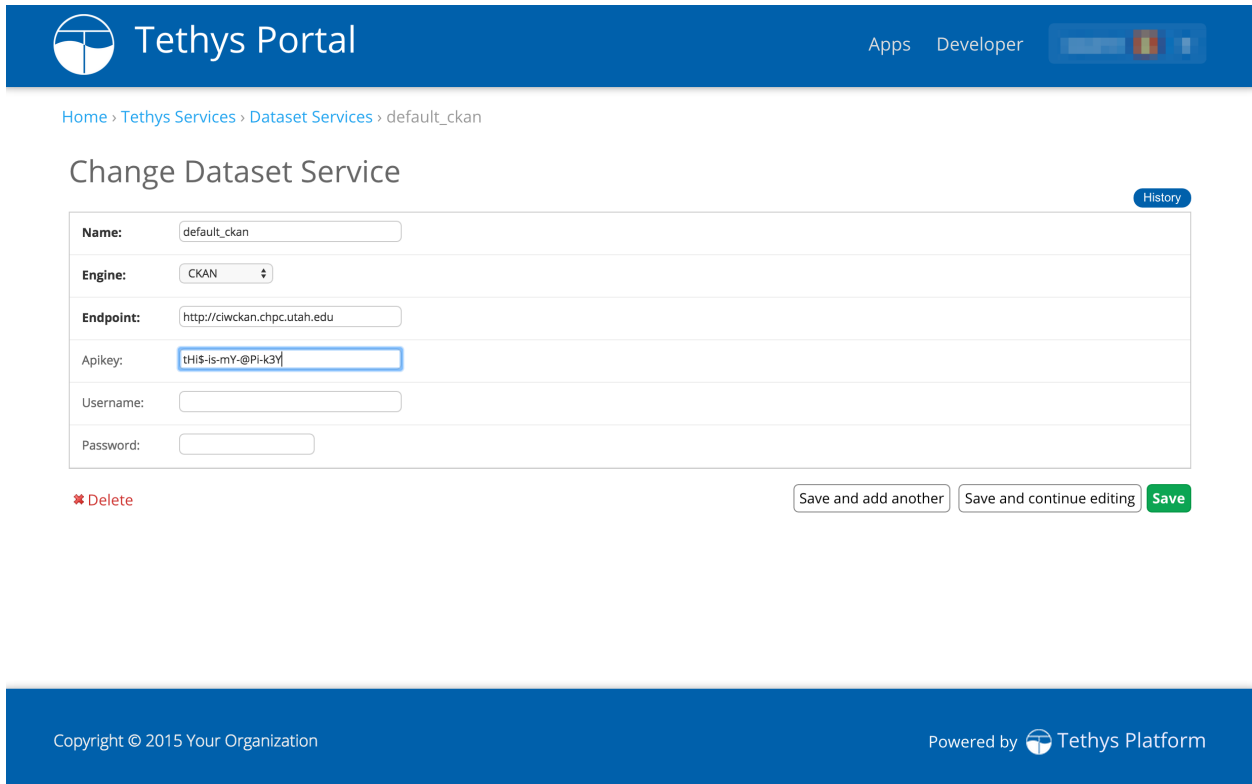


Figure 1.9: **Figure 5.** Fill out the form to register a new Dataset Service.

issued an API key. To access the API Key log into the CKAN site where you have an account and browse to your user profiles. The API key will be listed as a private attribute of your user profile.

5. Press “Save” to save the Dataset Service configuration.

What’s Next?

Head over to *Getting Started* and create your first app. You can also check out the *Software Development Kit* documentation to familiarize yourself with all the features that are available.

1.3.5 Update from 1.0 to 1.1

Last Updated: April 24, 2015

The following article describes how to update your Tethys Platform installation from version 1.0.X to 1.1.0. There are two methods that can be used to update to 1.1.0:

1. Delete Tethys Virtual Environment

Use the following command to delete the Tethys Platform virtual environment:

```
$ sudo rm -rf /usr/lib/tethys
```

2. Reinstall Tethys Platform

Follow the normal installation instructions to reinstall Tethys Platform, but do not update the Docker containers:

- *Installation on Linux*
- *Installation on Mac OSX*

Warning: Updating the Docker containers will result in the loss of all data.

3. Reinstall Apps

Reinstall any apps that you wish to have installed on Tethys Platform. Refer to the documentation for each app for specific installation instructions, but generally apps can be installed as follows:

```
$ . /usr/lib/tethys/bin/activate
(tethys) $ cd /path/to/tethysapp-my_first_app
(tethys) $ python setup.py install
(tethys) $ tethys syncstores my_first_app
```

The databases for the apps should have been retained unless you updated the Docker containers.

1.4 Tutorials

Last Updated: May 19, 2015

Use the following tutorials to learn the basics about Tethys Platform.

1.4.1 Getting Started

Last Updated: May 19, 2015

The getting started tutorial will walk you through the steps of setting up a new Tethys App project using Tethys Platform. If you have not already installed Tethys Platform, follow the [Installation](#) documentation and then return.

You will need to use the command line/terminal to manage your app and run the development server. It is highly recommended that you read the [Terminal Quick Guide](#) article for some tips if you are new to command line.

Create a New Tethys App Project

Last Updated: May 19, 2015

Tethys Platform provides an easy way to create new app projects called a scaffold. The scaffold generates a Tethys app project with the minimum files and the folder structure that is required (see [App Project Structure](#)). In this tutorial you will start a new Tethys app project using the scaffold and install it into your Tethys Platform ready for development.

Tip: You will need to use the command line/terminal to manage your app and run the development server. See the [Terminal Quick Guide](#) article for some tips if you are new to command line.

Generate Scaffold

To generate a new app using the scaffold, open a terminal, press CTRL-C to stop the development server if it is still running, and execute the following commands:

```
$ . /usr/lib/tethys/bin/activate
(tethys) $ mkdir ~/tethysdev
(tethys) $ cd ~/tethysdev
(tethys) $ tethys scaffold my_first_app
```

The final command from the code block above is provided by the Tethys [Command Line Interface](#). It will prompt you to enter metadata about your app such as, proper name, version, author, and description. All of these metadata are optional and you can accept the default value by pressing enter.

The commands you entered did the following tasks:

1. activated the Tethys *Python virtual environment*,
2. created a new directory called “tethysdev” in your home directory,
3. changed your working directory into the `tethysdev` directory, and
4. executed the **tethys scaffold** command to create the new app.

In a file browser change into your Home directory and open the `tethysdev` directory. If the scaffolding worked, you should see a directory called `tethysapp-my_first_app`. All of the source code for your app is located in this directory. Open the `tethysapp-my_first_app` and explore the contents. The main directory of your app project, `my_first_app`, is located within a namespace directory called `tethysapp`. Each part of the app project will be explained throughout these tutorials. For more information about the app project structure, see [App Project Structure](#).

Development Installation

Now that you have a new Tethys app project, you need to install the app into Tethys Platform. In a terminal, change into the `tethysapp-my_first_app` directory and execute the **python setup.py develop** command. Be sure to

activate the Tethys *Python virtual environment* if it is not already activated (see line 1 of the first code block):

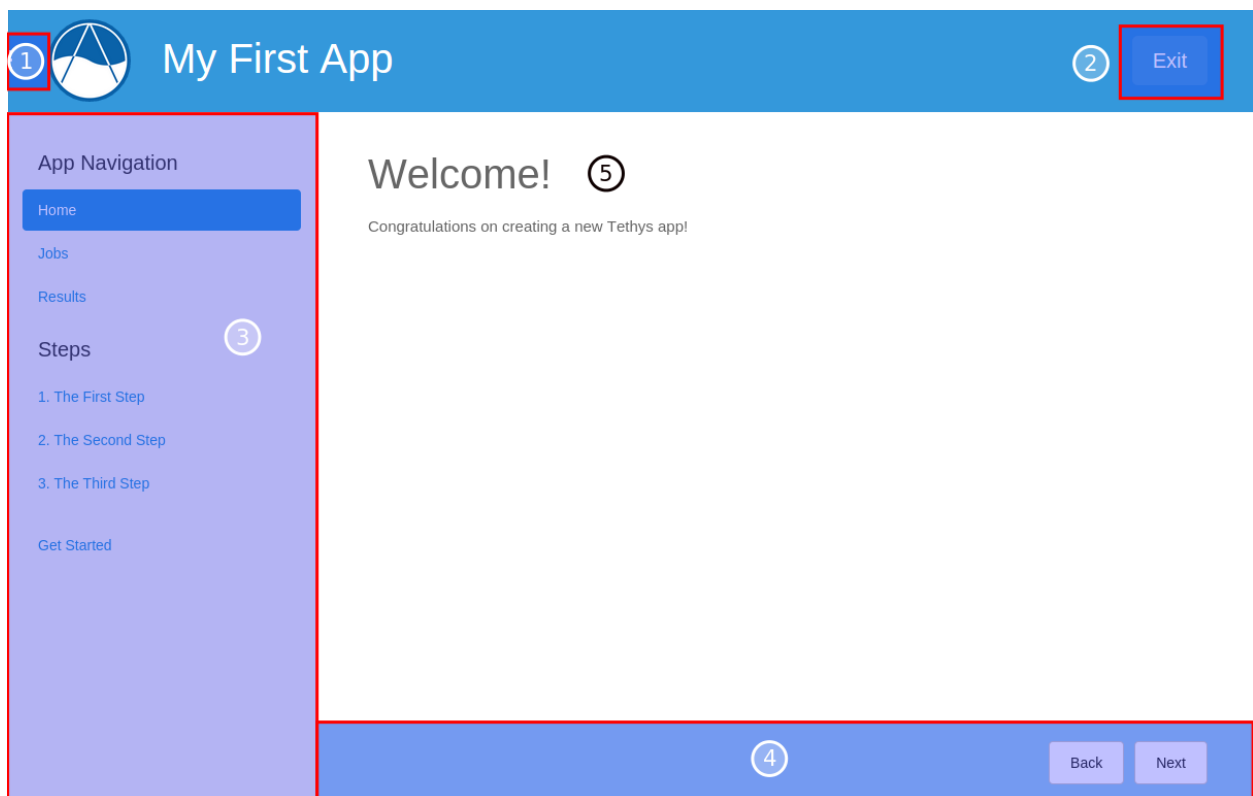
```
(tethys) $ cd ~/tethysdev/tethysapp-my_first_app
(tethys) $ python setup.py develop
```

View Your New App

Use start up the Tethys dockers and then the development server:

```
(tethys) $ tethys docker start
(tethys) $ tethys manage start
```

Browse to <http://127.0.0.1:8000/apps>. If all has gone well, you should see your app listed on the app library page. Exploring your new app won't take long, because there is only one page. Familiarize yourself with different parts of the app interface (see below).



Parts of a Tethys app interface: (1) app navigation toggle, (2) exit button, (3) app navigation, (4) actions, and (5) app content.

Tip: To stop the development server press CTRL-C. To stop the dockers run:

```
(tethys) $ tethys docker stop
```

Model View Controller

Tethys apps are developed using the *Model View Controller* (MVC) software architecture pattern. Following the MVC pattern will make your app project easier to develop and maintain in the future. Most of the code in your app will

fall into one of the three MVC categories. The Model represents the data of your app, the View is composed of the representation of the data, and the Controller consists of the logic to prepare the data for the view and any other logic your app needs. In the next few tutorials, you will be introduced to how the MVC development paradigm is used to develop Tethys apps. For more information about MVC, see *Key Concepts*.

App Project Paths

Throughout the tutorial, you will be asked to open various files. Most of the files will be located in your *app package* directory which shares the name of your app: “my_first_app”. If you generated your scaffold exactly as above, this directory will be located at the following path:

```
# Path to App Package Directory
~/tethysdev/tethysapp-my_first_app/tethysapp/my_first_app/
```

For convenience, all paths in the following tutorials will be given relative to the *app package* directory. For example:

```
# Relative App Package Directory Notation
my_first_app/controllers.py
```

Tip: As you explore the contents of your app project, you will notice that many of the directories have files named `__init__.py`. Though many of these files are empty, they are important and should not be deleted. They inform Python that the containing directory is a Python package. Python packages and their contents can be imported in Python scripts. Removing the `__init__.py` files could result in breaking import statements and it could make some of your code inaccessible. Similarly, if you add a directory to your project that contains Python modules and you would like them to be made available to your code, add a `__init__.py` file to the directory to make it a package.

The Model and Persistent Stores

Last Updated: May 20, 2015

In this part of the tutorial you’ll learn about the Model component of MVC development for Tethys apps. The Model represents the data of your app and the code used to manage it. The data of your app can take many forms. It can be generated on-the-fly and stored in Python data structures (e.g.: lists, dictionaries, and NumPy arrays), stored in databases, or contained in files via the *Dataset Services API*.

In this tutorial you will define your model using the *Persistent Stores API* to create a spatially enabled database for your app and you will learn how to use the SQLAlchemy object relational mapper (ORM) to create a data model for your app.

Register a Persistent Store

The Tethys Portal provides the *Persistent Stores API* to streamline the use of SQL databases in apps. To register a new *persistent store* database add the `persistent_stores()` method to your *app class*, which is located in your *app configuration file*. This method must return a list or tuple of `PersistentStore` objects.

Open the app configuration file for your app located at `my_first_app/app.py` in your favorite text editor. Import the `PersistentStore` object at the top of the file, add the `persistent_stores()` method to your app class, and save the changes:

```
from tethys_apps.base import TethysAppBase, url_map_maker
from tethys_apps.base import PersistentStore
```

```
class MyFirstApp(TethysAppBase):
```

```

"""
Tethys App Class for My First App.
"""

name = 'My First App'
index = 'my_first_app:home'
icon = 'my_first_app/images/icon.gif'
package = 'my_first_app'
root_url = 'my-first-app'
color = '#3498db'

def url_maps(self):
    """
    Add controllers
    """
    UrlMap = url_map_maker(self.root_url)

    url_maps = (UrlMap(name='home',
                       url='my-first-app',
                       controller='my_first_app.controllers.home'
                       ),
               )

    return url_maps

def persistent_stores(self):
    """
    Add one or more persistent stores
    """
    stores = (PersistentStore(name='stream_gage_db',
                              initializer='init_stores:init_stream_gage_db',
                              spatial=True
                              ),
            )

    return stores

```

A persistent store database will be created for each `PersistentStore` object that is returned by the `persistent_stores()` method of your *app class*. In this case, your app will have a persistent store named “stream_gage_db”. The `initializer` argument points to a function that you will define in a later step. The `spatial` argument can be used to add spatial capabilities to your persistent store. Tethys Platform provides PostgreSQL databases for persistent stores and PostGIS for the spatial database capabilities.

Note: Read more about persistent stores in the *Persistent Stores API* documentation.

Create an SQLAlchemy Data Model

After your database is created, you will need to create the tables that will store the data for your app. The plan for your database tables or schema is called a data model. SQLAlchemy provides an Object Relational Mapper (ORM) that allows you to create data models using Python code and issue queries using an object-oriented approach. In other words, you are able to harness the power of SQL databases without writing SQL. As a primer to SQLAlchemy ORM, we highly recommend you complete the [Object Relational Tutorial](#).

In this step, you will use SQLAlchemy to create a data model for the tables that will store the data for your app. Open the `model.py` file located at `my_first_app/model.py` in a text editor.

First, add the following import statements to your `model.py` file:

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, Float
from sqlalchemy.orm import sessionmaker

from .utilities import get_persistent_store_engine
```

Next, add these lines to your `model.py` file:

```
# DB Engine, sessionmaker and base
engine = get_persistent_store_engine('stream_gage_db')
SessionMaker = sessionmaker(bind=engine)
Base = declarative_base()
```

The `get_persistent_store_engine()` method that is used here accepts the name of a persistent store as an argument and returns an SQLAlchemy engine object. The engine object contains the connection information needed to connect to the persistent store database. Anytime you want to query or modify your persistent store data, you will do so with an SQLAlchemy session object. As the name implies, the `SessionMaker` can be used to create new session objects. The `Base` object is used in the next step when we define our data model. Add these lines to your `model.py` file:

```
# SQLAlchemy ORM definition for the stream_gages table
class StreamGage(Base):
    """
    Example SQLAlchemy DB Model
    """
    __tablename__ = 'stream_gages'

    # Columns
    id = Column(Integer, primary_key=True)
    latitude = Column(Float)
    longitude = Column(Float)
    value = Column(Integer)

    def __init__(self, latitude, longitude, value):
        """
        Constructor for a gage
        """
        self.latitude = latitude
        self.longitude = longitude
        self.value = value
```

Each class in an SQLAlchemy data model defines a table in the database. The model you defined above consists of a single table called “stream_gages”, as denoted by the `__tablename__` property of the `StreamGage` class. The `StreamGage` class inherits from the `Base` class that we created in the previous lines. This inheritance notifies SQLAlchemy that the `StreamGage` class is part of the data model.

The class defines four other properties that are SQLAlchemy `Column` objects: `id`, `latitude`, `longitude`, and `value`. These properties define the columns of the “stream_gages” table. The column type and options are defined by the arguments passed to the `Column` constructor. For example, the `latitude` column is of type `Float` while the `id` column is of type `Integer` and is also flagged as the primary key for the table. The `StreamGage` class also has a simple constructor method called `__init__()`.

This class is not only used to define the tables for your persistent store, it will also be used to create objects for interacting with your data.

Be sure to save the changes to `model.py` and close before proceeding.

Create an Initialization Function

Now that you have created a data model, the next step is to write a database initialization function. This function will be called during the initialization phase of your persistent store database and will be used to create the tables in your database and add any initial data that you may need in the database for your app to work.

Open the `my_first_app/init_stores.py` in a text editor. At the top of this file, import the engine, `SessionMaker`, `Base`, and `StreamGage` from your data model:

```
from .model import engine, SessionMaker, Base, StreamGage
```

Next, create a new function called `init_stream_gage_db()` with a single argument called `first_time` and the following code:

```
def init_stream_gage_db(first_time):
    """
    An example persistent store initializer function
    """
    # Create tables
    Base.metadata.create_all(engine)

    # Initial data
    if first_time:
        # Make session
        session = SessionMaker()

        # Gage 1
        gage1 = StreamGage(latitude=40.23812952992122,
                           longitude=-111.69585227966309,
                           value=1)

        session.add(gage1)

        # Gage 2
        gage2 = StreamGage(latitude=40.238784729316215,
                           longitude=-111.7101001739502,
                           value=2)

        session.add(gage2)

        # Gage 3
        gage3 = StreamGage(latitude=40.23650788415366,
                           longitude=-111.73278093338013,
                           value=3)

        session.add(gage3)

        # Gage 4
        gage4 = StreamGage(latitude=40.242519244799816,
                           longitude=-111.68254852294922,
                           value=4)

        session.add(gage4)

    session.commit()
```

The `Base.metadata.create_all(engine)` line is all that is needed to create the tables in your persistent store database. Every class that inherits from the `Base` class is tracked by a metadata object. The

`metadata.create_all()` method issues the SQL that is needed to create the tables associated with the `Base` class. Notice that you must give it the `engine` object for connection information.

The `first_time` parameter that is passed to all persistent store initialization functions is a boolean that is `True` if the function is being called after the tables have been created for the first time. This is provided as a mechanism for adding initial data only the first time. Notice the code that adds initial data to your persistent store database is wrapped in a conditional statement that uses the `first_time` parameter.

This initial data code adds four stream gages to your persistent store database. Creating a new record in the database using SQLAlchemy is achieved by creating a new `StreamGage` object and adding it to the `session` object using the `session.add()` method. To persist the new records to the persistent store database, the `session.commit()` method is called. You will learn how to query the persistent store database using SQLAlchemy in the *The Controller* tutorial.

Save your changes to `init_stores.py` and close before moving on.

Register Initialization Function

Recall that when you registered the persistent store in your app configuration file, you specified the initializer function for the persistent store. This argument accepts a string representing the path to the function using dot notation and a colon to delineate the function (e.g.: “foo.bar:function”). Check your *app configuration file* (`app.py`) to ensure the path to the initializer function is correct: `'init_stores:init_stream_gage_db'`.

Persistent Store Initialization

The Tethys command line utility provides a command for initializing persistent stores. Save all changes to the files you edited and stop your development server using `CTRL-C` if it is still running. It is possible that your server may have crashed during editing and is displaying errors; ignore these errors. Execute the following command in the terminal:

```
(tethys) $ tethys syncstores my_first_app
```

The database will be initialized and you will see text printed to the terminal that will indicate this:

```
Loading Tethys Apps...
Tethys Apps Loaded: my_first_app

Provisioning Persistent Stores...
Creating database "stream_gage_db" for app "my_first_app"...
Enabling PostGIS on database "stream_gage_db" for app "my_first_app"...
Initializing database "stream_gage_db" for app "my_first_app" using initializer
"init_stream_gage_db"...
```

If you have a graphical database client like *PGAdmin III*, you may wish to connect to your PostgreSQL database server and confirm that the database was created. You can use the credentials for `tethys_super` database user that you defined during installation to connect to the database. The name of the database will be a combination of the name of your app and the name of the persistent store: (e.g.: `my_first_app_stream_gage_db`). For a more detailed explanation of connecting to your database using *PGAdmin III*, see the *PGAdmin III Tutorial*.

Example of graphical database client PGAdmin III.

The View and Templating

Last Updated: May 20, 2015

In this section the View aspect of MVC will be introduced. The View consists of the representation or visualizations of your app’s data and the user interface. Views for Tethys apps are constructed using the standard web programming tools: HTML, JavaScript, and CSS. Additionally, Tethys Platform provides the Django Python templating language

The screenshot shows the pgAdmin III interface. On the left, the 'Object browser' tree is expanded to show the database 'my_first_app_stream_gage_db' under the server 'tethys (localhost:5432)'. The main pane displays the 'Properties' tab for this database, showing a list of properties and their values. Below the properties, the 'SQL pane' contains the SQL command used to create the database.

Property	Value
Name	my_first_app_stream_gage_db
OID	21827
Owner	tethys_db_manager
ACL	
Tablespace	pg_default
Default tablespace	pg_default
Encoding	UTF8
Collation	en_US.UTF-8
Character type	en_US.UTF-8
Default schema	public
Default table ACL	
Default sequence ACL	
Default function ACL	
Default type ACL	
Allow connections?	Yes
Connected?	Yes
Connection limit	-1
System database?	No
Comment	

```
-- Database: my_first_app_stream_gage_db
-- DROP DATABASE my_first_app_stream_gage_db;

CREATE DATABASE my_first_app_stream_gage_db
  WITH OWNER = tethys_db_manager
  ENCODING = 'UTF8'
  TABLESPACE = pg_default
  LC_COLLATE = 'en_US.UTF-8'
  LC_CTYPE = 'en_US.UTF-8'
  CONNECTION LIMIT = -1;
```

Retrieving details on database my_first_app_stream_gage_db... Done. 0.00 secs

allowing you to insert Python code into your HTML documents, similar to how PHP is used. The result is dynamic, reusable templates for the web pages of your app.

In this tutorial you will add a view to your app for displaying the stream gages that are in your database on a Google Map.

Templating

The Django template language is a simple, but powerful templating language. This section will provide a crash course in Django template language basics, but we highly recommend a review of the [Django Template Language](#) documentation.

Browse to your templates directory located at `my_first_app/templates/`. By convention, all the templates for your app are stored in a directory with the same name of your *app package* inside the templates directory (e.g.: `templates/my_first_app`). This will prevent potential conflicts with the templates of other apps. You will find two templates in this directory: `base.html` and `home.html`. Refer to these templates as the Django template concepts are introduced.

Variables, Filters, and Tags Django templates can contain variables, filters, and tags. Variables are denoted by double curly brace syntax like this: `{{ variable }}`. Template variables are replaced by the value of the variable. Dot notation can be used access attributes of a variable: `{{ variable.attribute }}`.

Variables can be modified by filters which look like this: `{{ variable|filter:argument }}`. Filters perform modifying functions on variable output such as formatting dates, formatting numbers, changing the letter case, and concatenating multiple variables.

Tags use curly-brace-percent-sign syntax like this: `{% tag %}`. Tags perform many different functions including creating text, controlling flow, or loading external information to be used in the app. Some commonly used tags include `for`, `if`, `block`, and `extends`.

Tip: For a better explanation of variables, filters and tags, see the [App Templating API](#).

Template Inheritance One of the advantages of using the Django template language is that it provides a way for child templates to extend parent templates, which reduces the amount of HTML you need to write. Template inheritance is accomplished using two tags: `extends` and `block`. Parent templates provide `blocks` of content that can be overridden by child templates. Child templates can extend parent templates by using the `extends` tag and specifying the template they which to inherit from. Calling the `block` tag of a parent template in a child template will override any content in that `block` tag with the content in the child template.

Tip: If you are unfamiliar with Django template inheritance, please review the [Django Template Inheritance](#) documentation before proceeding.

Base Template Tethys apps generated from the scaffold come with a `base.html` template which has the following contents:

```
{% extends "tethys_apps/app_base.html" %}

{% load staticfiles %}

{% block title %}- {{ tethys_app.name }}{% endblock %}

{% block styles %}
    {{ block.super }}
{% endblock %}
```

```

<link href="{% static 'my_first_app/css/main.css' %}" rel="stylesheet"/>
{% endblock %}

{% block app_icon %}
    {# The path you provided in your app.py is accessible through the tethys_app.icon
    context variable 
{% endblock %}

{# The name you provided in your app.py is accessible through the tethys_app.name
context variable #}{% block app_title %}{{ tethys_app.name }}{% endblock %}

{% block app_navigation_items %}
    <li class="title">App Navigation</li>
    <li class="active"><a href="">Home</a></li>
    <li><a href="">Jobs</a></li>
    <li><a href="">Results</a></li>
    <li class="title">Steps</li>
    <li><a href="">1. The First Step</a></li>
    <li><a href="">2. The Second Step</a></li>
    <li><a href="">3. The Third Step</a></li>
    <li class="separator"></li>
    <li><a href="">Get Started</a></li>
{% endblock %}

{% block app_content %}
{% endblock %}

{% block app_actions %}
{% endblock %}

{% block scripts %}
    {{ block.super }}
    <script src="{% static 'my_first_app/js/main.js' %}" type="text/javascript"></script>
{% endblock %}

```

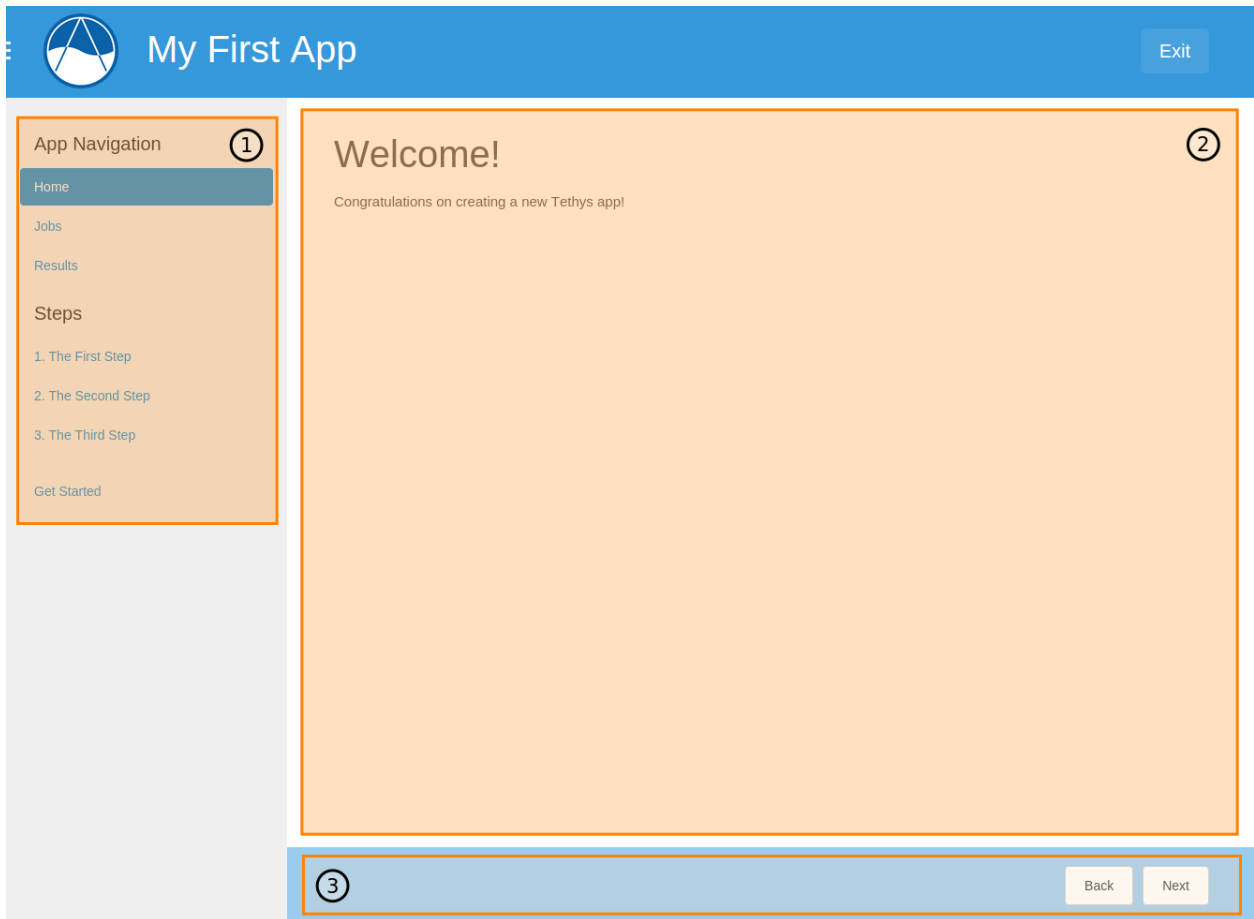
The `base.html` template is intended to be used as the parent template for all your app templates via the `extends` tag. It contains several `block` tags that your app templates can override or extend. The `block` tags you will use most often are `app_navigation_items`, `app_content`, and `app_actions`. These blocks correspond with different parts of the app interface (shown in the figure below). As a rule, content that you would like to be present in all your templates should be included in the `base.html` template and content that is specific to a certain template should be included in that template.

The `block` tags of the `base.html` template correspond with different parts of the interface: (1) `app_navigation_items`, (2) `app_content`, and (3) `app_actions`.

Tip: For an explanation of the blocks in the `base.html` template see the *App Templating API*.

Public Files and Resources

Most apps will use files and resources that are static—meaning they do not need to be preprocessed before being served like templates do. Examples of these files include images, CSS files, and JavaScript files. Tethys Platform will automatically register static files that are located in the `public` directory of your app project. Use the `static` tag in templates to load the resource URLs. The `base.html` template provides examples of how to use the `static` tag. See the Django documentation for the `static` tag for more details.



Caution: Any file stored in the public directory will be accessible to anyone. Be careful not to expose sensitive information.

Make a New Template

Now that you know the basics of templating, you will learn how to create new templates that extend the base template and use the `block` tags. Create a new template in your templates directory (`my_first_app/templates/my_first_app/`) and name it `map.html`. Open this file in a text editor and copy and paste the following code into it:

```
{% extends "my_first_app/base.html" %}

{% load tethys_gizmos %}

{% block app_content %}
    <h1>Stream Gages</h1>
    {% gizmo map_view map_options %}
{% endblock %}

{% block app_actions %}
    <a href="{% url 'my_first_app:home' %}" class="btn btn-default">Back</a>
{% endblock %}
```

The `map.html` template that you created extends the `base.html` template. It also overrides the `app_content`, `app_actions`, and `scripts` blocks. An action called “Back” is added to the `app_actions` block. It uses a new tag, the `url` tag, to provide a link back to the home page of the app. The `url` tag will be discussed in more detail in the [URL Mapping](#) tutorial.

The map is inserted into the `app_content` block using one of the Tethys Gizmos called `map_view`. Gizmos are an easy way to insert common user interface elements in to your templates with minimal code. The map is configured via a dictionary called `map_options`, which is defined in the controller. This will be discussed in the next tutorial. For more information on Gizmos, refer to the [Template Gizmos API](#) documentation.

The Controller

Last Updated: May 20, 2015

The Controller component of MVC will be discussed in this part of the tutorial. The job of the controller is to coordinate between the View and the Model. Often this means querying a database and transforming the data to a format that the view expects it to be in. The Controller also handles most of the application logic such as processing and validating form data or launching model runs. In a Tethys app, controllers are simple Python functions.

Django is used to implement Tethys controllers but they are called “views” in Django. The [Writing Views](#) documentation for Django is a good reference for Tethys controllers. Note that URL mapping is handled differently in Tethys app development than in Django development and will be discussed in the [URL Mapping](#) tutorial.

In this tutorial you will write a controller that will retrieve the data from your stream gage model and then pass it to the template that you created in the previous tutorial.

Make a New Controller

Recall that in [The Model and Persistent Stores](#) tutorial you created an SQLAlchemy data model to store information about stream gages. You also created an initialization function that loaded some dummy data into your database. You will now add some logic to your controller to retrieve this data and pass it to the template.

Open your `controllers.py` file located at `my_first_app/controllers.py`. This file should contain a function called `home`. This function is the controller for the home page of your app. All controller functions must accept a request object and they must return a response object. The request object contains information about the HTTP request, including any form data that is submitted (more on this later). There are several ways to return a response object, but the most common way is to use the `render()` function provided by Django. This function requires three arguments: the request object, the template to be rendered, and the context dictionary.

Add the following imports to the top of the file:

```
from .model import SessionMaker, StreamGage
from tethys_gizmos.gizmo_options import MapView, MVLayer, MVView
```

Then add a new controller function called `map` after the `home` function:

```
def map(request):
    """
    Controller for map page.
    """
    # Create a session
    session = SessionMaker()

    # Query DB for gage objects
    gages = session.query(StreamGage).all()

    # Transform into GeoJSON format
    features = []

    for gage in gages:
        gage_feature = {
            'type': 'Feature',
            'geometry': {
                'type': 'Point',
                'coordinates': [gage.longitude, gage.latitude]
            }
        }

        features.append(gage_feature)

    geojson_gages = {
        'type': 'FeatureCollection',
        'crs': {
            'type': 'name',
            'properties': {
                'name': 'EPSG:4326'
            }
        },
        'features': features
    }

    # Define layer for Map View
    geojson_layer = MVLayer(source='GeoJSON',
                            options=geojson_gages,
                            legend_title='Provo Stream Gages',
                            legend_extent=[-111.74, 40.22, -111.67, 40.25])

    # Define initial view for Map View
    view_options = MVView(
        projection='EPSG:4326',
        center=[-100, 40],
```

```

        zoom=3.5,
        maxZoom=18,
        minZoom=2
    )

    # Configure the map
    map_options = MapView(height='500px',
                          width='100%',
                          layers=[geojson_layer],
                          view=view_options,
                          basemap='OpenStreetMap',
                          legend=True)

    # Pass variables to the template via the context dictionary
    context = {'map_options': map_options}

    return render(request, 'my_first_app/map.html', context)

```

The new map controller queries the persistent store for the stream gages, converts the data into [GeoJSON](#) format for the map, and configures the map options for the Map View Gizmo that is used in the template.

To query the database, an `SQLAlchemy` `session` object is needed. It is created using the `SessionMaker` object imported from the `model.py` file. Querying is accomplished by using the `query()` method on the `session` object. The result is a list of `StreamGage` objects representing the records in the database.

The map is capable of consuming spatial data in a few formats including GeoJSON, so the map controller handles the job of converting the data from the list of `StreamGage` objects to GeoJSON format.

The map Gizmo that is used in the `map.html` template requires a dictionary of configuration options called “`map_options`”. This is created in the controller and the `input_overlays` option is used to give the GeoJSON formatted stream gage data to the map.

Next, a template context dictionary is defined that contains all of the variables that you wish to be available for use in the template.

Finally, the `render()` function is used to create the response object. It is in the `render()` function that you specify the template that is to be rendered by the controller. In this case, the `map.html` that you created in the last tutorial. Note that the path you provide to the template is relative to the template directory of your app: `my_first_app/map.html`.

Save `controllers.py` before going on.

URL Mapping

Last Updated: May 20, 2015

Whenever you create a new controller, you will also need to associate it with a URL by creating URL map for it. When a URL is requested, the controller that it is mapped to will be executed.

In this tutorial you will create a new URL map for the new `map` controller you created in the previous tutorial.

Map Controller to URL

Mapping a controller to a URL is performed in the *app configuration file* (`app.py`). Open your app configuration file located at `my_first_app/app.py`. Your *app class* will already have a method called `url_maps()`. This method must return a list or tuple of `UrlMap` objects. `UrlMap` objects require three attributes: `name`, `url`, and `controller`.

Your `app` class will already have one `UrlMap` for the home page called “home”. Add a new `UrlMap` object for the map controller that you created in the previous step. Give it a name of “map”, a url of “my-first-app/map”, and a path to the controller of “my_first_app.controllers.map”. The `url_maps()` method for your app should look something like this when you are done:

```
class MyFirstApp(TethysAppBase):
    """
    Tethys App Class for My First App.
    """
    name = 'My First App'
    index = 'my_first_app:home'
    icon = 'my_first_app/images/icon.gif'
    package = 'my_first_app'
    root_url = 'my-first-app'
    color = '#3498db'

    def url_maps(self):
        """
        Add controllers
        """
        UrlMap = url_map_maker(self.root_url)

        url_maps = (UrlMap(name='home',
                           url='my-first-app',
                           controller='my_first_app.controllers.home'),
                   UrlMap(name='map',
                           url='my-first-app/map',
                           controller='my_first_app.controllers.map'),
                  )

        return url_maps

    def persistent_stores(self):
        """
        Add one or more persistent stores
        """
        stores = (PersistentStore(name='stream_gage_db',
                                  initializer='init_stores:init_stream_gage_db',
                                  spatial=True
                                ),
                 )

        return stores
```

Important: All of the URL patterns for your app should begin with the base URL of your app (e.g.: ‘my-first-app’) to prevent conflicts with other apps.

Now that you have created the URL map for your new map page, you can create a link to it from the home page. Open the `home.html` template located at `my_first_app/templates/my_first_app/home.html`. Replace the `app_actions` template block with the following:

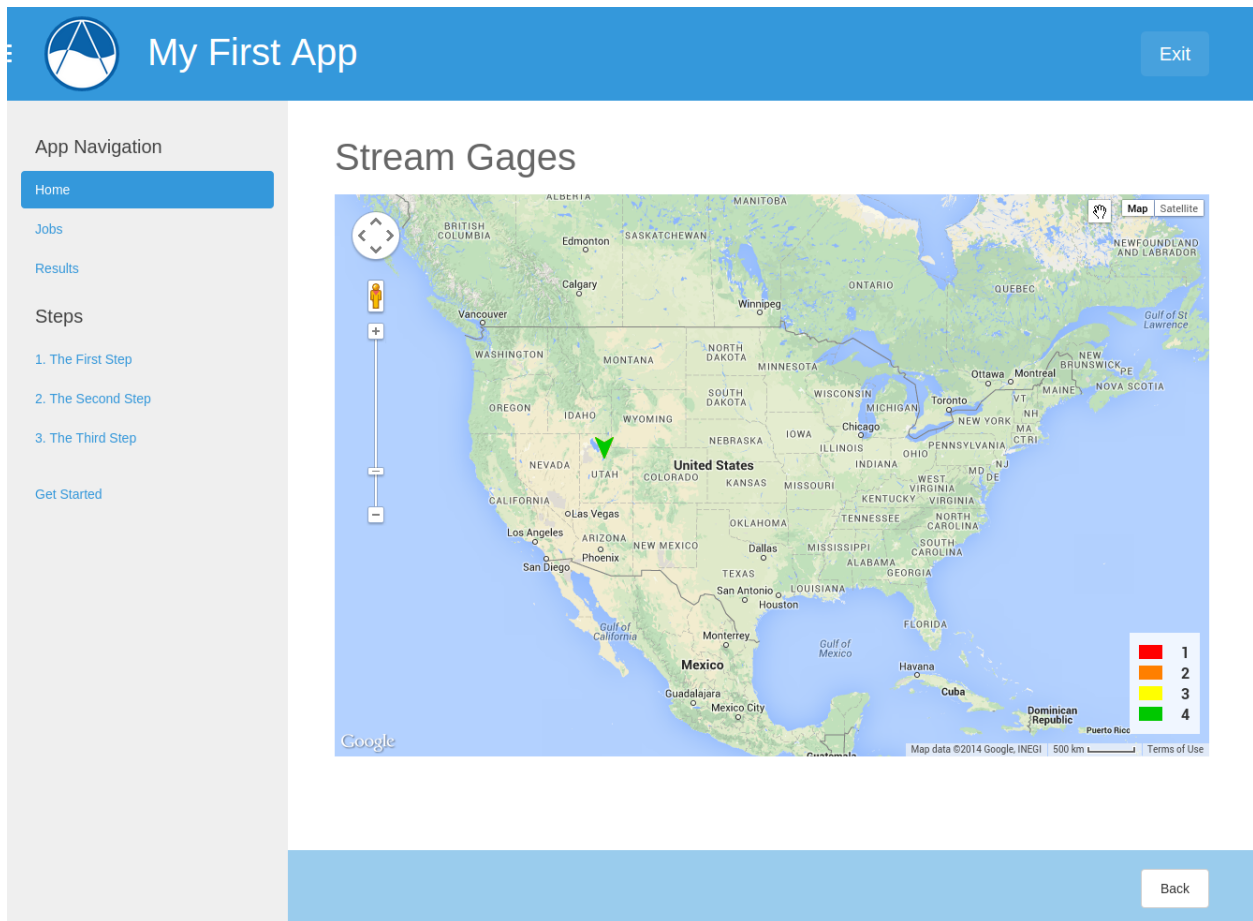
```
{% block app_actions %}
  <a href="{% url 'my_first_app:map' %}" class="btn btn-default">Go To Map</a>
{% endblock %}
```

In this code, the `url` template tag is used to provide the url to the map page. It accepts a string with the following pattern: “name_of_app:name_of_url_map”. The advantage of using the `url` tag as opposed to hard coding the URL is that if the URL ever needs to be changed, you will only need to change it in your app configuration file and

not in every template that references that URL.

View New Map Page

At this point, your app should be ready to run again. Save all changes in the files you edited and restart the development server using the `tethys manage start` command in the terminal (stop it using `CTRL-C` if necessary). Browse to your app home page at <http://127.0.0.1:8000/apps/my-first-app>. Use the “Go To Map” action to browse to your new map page. It should look similar to this:



Advanced Concepts

Last Updated: May 20, 2015

The purpose of this tutorial will be to introduce some advanced concepts in Tethys app development. In the map page you created in the previous tutorials, you are able to view all of the stream gages on a map concurrently. In this tutorial you will add the ability to view individual stream gages on the map page. This will involve creating a new url map, new controller, and some modifications to the map template. This exercise will also serve as a good review of MVC development in Tethys Platform.

New URL Map and URL Variables

You can add variables to your URLs to make your controllers and web pages more dynamic. URL variables are denoted by single curly braces in the URL string like this: `/example/url/{variable}`. Open the `my_first_app/app.py` file in a text editor. Modify the `url_maps()` method by adding a new `UrlMap` object named “`map_single`” with a URL variable called “`id`”. Your `url_maps()` method should look like this when you are done:

```
class MyFirstApp(TethysAppBase):
    """
    Tethys App Class for My First App.
    """

    name = 'My First App'
    index = 'my_first_app:home'
    icon = 'my_first_app/images/icon.gif'
    package = 'my_first_app'
    root_url = 'my-first-app'
    color = '#3498db'

    def url_maps(self):
        """
        Add controllers
        """
        UrlMap = url_map_maker(self.root_url)

        url_maps = (UrlMap(name='home',
                          url='my-first-app',
                          controller='my_first_app.controllers.home'),
                   UrlMap(name='map',
                          url='my-first-app/map',
                          controller='my_first_app.controllers.map'),
                   UrlMap(name='map_single',
                          url='my-first-app/map/{id}',
                          controller='my_first_app.controllers.map_single'),
                  )

        return url_maps

    def persistent_stores(self):
        """
        Add one or more persistent stores
        """
        stores = (PersistentStore(name='stream_gage_db',
                                 initializer='init_stores:init_stream_gage_db',
                                 spatial=True),
                 ),

        return stores
```

Note: The Django documentation on URL mapping will not be useful for Tethys apps. A different approach is used by Tethys that is easier to use than the Django method.

New Controller

Notice that the `map_single` `UrlMap` object points to a controller named “`map_single`”. This controller doesn’t exist yet, so we will need to create it. Open the `my_first_app/controllers.py` in a text editor and add the `map_single` controller function to it:

```
def map_single(request, id):
    """
    Controller for map page.
    """
    # Create a session
    session = SessionMaker()

    # Query DB for gage objects
    gage = session.query(StreamGage).filter(StreamGage.id==id).one()

    # Transform into GeoJSON format
    gage_feature = {
        'type': 'Feature',
        'geometry': {
            'type': 'Point',
            'coordinates': [gage.longitude, gage.latitude]
        }
    }

    geojson_gages = {
        'type': 'FeatureCollection',
        'crs': {
            'type': 'name',
            'properties': {
                'name': 'EPSG:4326'
            }
        },
        'features': [gage_feature]
    }

    # Define layer for Map View
    geojson_layer = MVLayer(source='GeoJSON',
                           options=geojson_gages,
                           legend_title='Provo Stream Gages',
                           legend_extent=[-111.74, 40.22, -111.67, 40.25])

    # Define initial view for Map View
    view_options = MVView(
        projection='EPSG:4326',
        center=[-111.70, 40.24],
        zoom=13,
        maxZoom=18,
        minZoom=2
    )

    # Configure the map
    map_options = MapView(height='500px',
                          width='100%',
                          layers=[geojson_layer],
                          view=view_options,
                          basemap='OpenStreetMap',
                          legend=True)
```



```
context = {'map_options': map_options,
          'gage_id': id}

return render(request, 'my_first_app/map.html', context)
```

The `map_single` controller function is slightly different than the `map` controller you created earlier. It accepts an additional argument called “`id`”. The `id` URL variable value will be passed to the `map_single` controller making the `id` variable available for use in the controller logic.

Anytime you create a URL with variables in it, the variables need to be added to the arguments of the controller function it maps to.

The `map_single` controller is similar but different from the `map` controller you created earlier. The SQLAlchemy query searches for a single stream gage record using the `id` variable via the “`filter()`” method. The stream gage data returned by the query is reformatted into GeoJSON format as before and the `map_options` for the Gizmo are defined.

The context is expanded to include the `id` variable, so that it will be available for use in the template. The same `map.html` template is being used by this controller as was used by the `map` controller. However, it will need to be modified slightly to make use of the new `gage_id` context variable.

Modify the Template

Open the `map.html` template located at `my_first_app/templates/my_first_app/map.html`. Modify the template so that it matches this:

```
{% extends "my_first_app/base.html" %}

{% load tethys_gizmos %}

{% block app_navigation_items %}
  <li class="title">Gages</li>
  <li{% if not gage_id %} class="active"{% endif %}>
    <a href="{% url 'my_first_app:map' %}">All Gages</a>
  </li>
  <li{% if gage_id == '1' %} class="active"{% endif %}>
    <a href="{% url 'my_first_app:map_single' id=1 %}">Stream Gage 1</a>
  </li>
  <li{% if gage_id == '2' %} class="active"{% endif %}>
    <a href="{% url 'my_first_app:map_single' id=2 %}">Stream Gage 2</a>
  </li>
  <li{% if gage_id == '3' %} class="active"{% endif %}>
    <a href="{% url 'my_first_app:map_single' id=3 %}">Stream Gage 3</a>
  </li>
  <li{% if gage_id == '4' %} class="active"{% endif %}>
    <a href="{% url 'my_first_app:map_single' id=4 %}">Stream Gage 4</a>
  </li>
{% endblock %}

{% block app_content %}
  {% if gage_id %}
    <h1>Stream Gage {{gage_id}}</h1>
  {% else %}
    <h1>Stream Gages</h1>
  {% endif %}

  {% gizmo map_view map_options %}
```

```
{% endblock %}

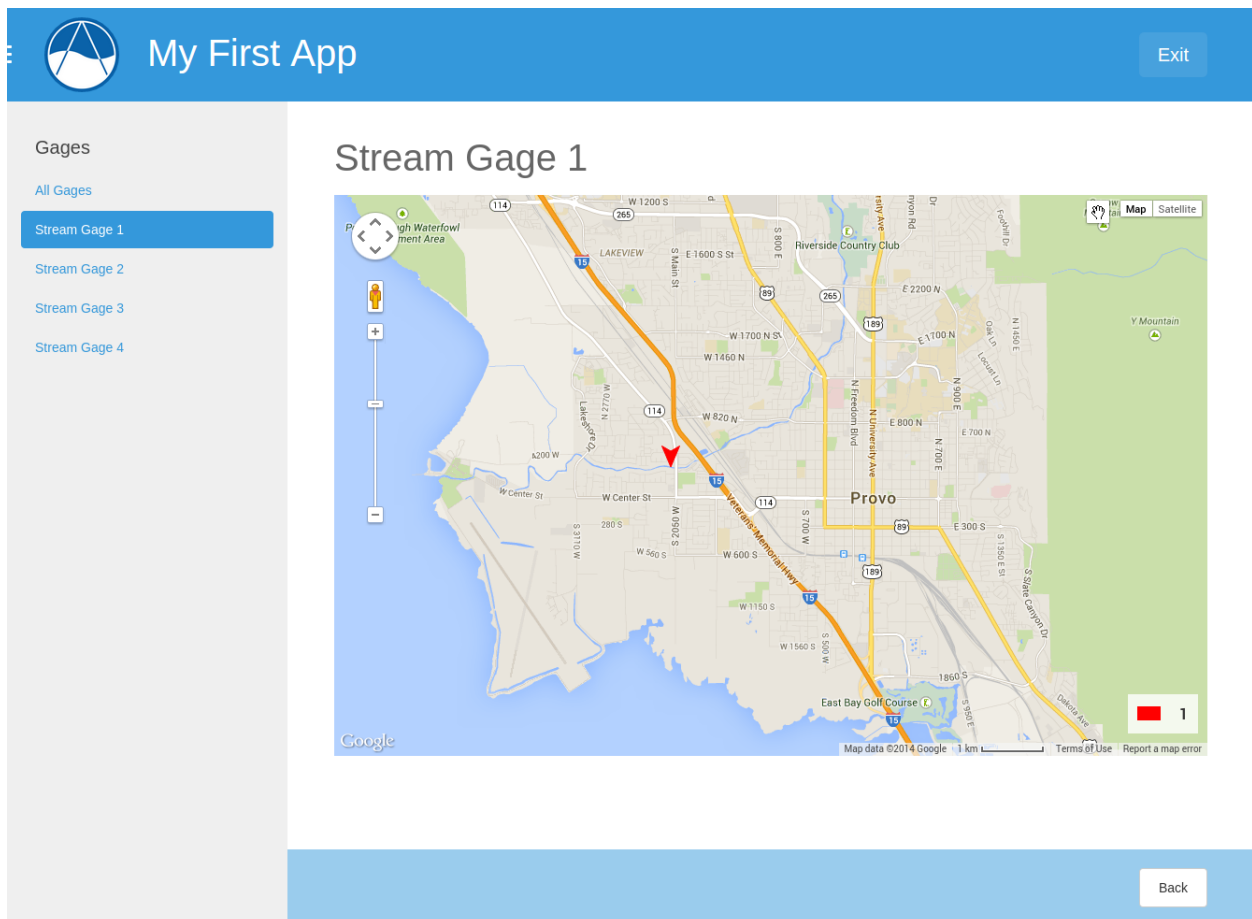
{% block app_actions %}
  <a href="{% url 'my_first_app:home' %}" class="btn btn-default">Back</a>
{% endblock %}
```

There are two changes to the `map.html` template that are worth noting. First, the template now overrides the `app_navigation_block` to provide links for each of the stream gages in the navigation. The `if` template tag is used in each of the nav items to highlight the appropriate link based on the `gage_id`. Notice that all `if` tags must also end with a `endif` tag. The text between the two tags is displayed only if the conditional statement evaluates to `True`. The `href` for each link is provided using the `url`, but this time the `id` variable is also provided as an argument.

The other change to the template is the heading of the page (`<h1>`) is wrapped by `if`, `else`, and `endif` tags. The effect is to display “Stream Gage id#” when viewing only one stream gage and “Stream Gages” when viewing all of them.

View Updated Map Page

Just like that, you added a new page to your app using MVC. Save the changes to any files you edited and start up the development server using the `tethys manage start` command and browse to your app. Use the “Go To Map” action on the home page to browse to your new map page and use the options in the navigation pane to view the different gages. It should look like this (although you may need to pan and zoom some):



Variable URLs

Take note of the URL as you are viewing the different gages. You should see the ID of the current gage. For example, the URL for the gage with an ID of 1 would be <http://127.0.0.1:8000/apps/my-first-app/map/1/>. You can manually change the ID in the URL to request the gage with that ID. Visit this URL <http://127.0.0.1:8000/apps/my-first-app/map/3/> and it will map the gage with ID 3.

Try this URL: <http://127.0.0.1:8000/apps/my-first-app/map/100/>. You should see a lovely error message, because you don't have a gage with ID 100 in the database. This uncovers a bug in your code that we won't take the time to fix in this tutorial. If this were a real app, you would need to handle the case when the ID doesn't match anything in the database so that it doesn't give you an error.

This exercise also exposes a vulnerability with using integer IDs in the URL—they can be guessed easily. For example if your app had a delete method, it would be very easy for an attacker to write a script that would increment through integers and call the delete method—effectively clearing your database. It would be a much better practice to use a UUID (see [Universally unique identifier](#)) or something similar for IDs.

User Input and Forms

Last Updated: May 20, 2015

Eventually you will need to request input from the user, which will involve working with HTML forms. In this tutorial, you'll learn how to create forms in your template and process the data submitted through the form in your controller.

New URL Map

The form will be created on a new page, which means you will need to create a new URL map and controller. Open your `my_first_app/app.py` and add a new `UrlMap` object called “echo_name” to the `url_maps()` method of your `app class`. The `url_maps()` method of your app class should look like this now:

```
class MyFirstApp(TethysAppBase):
    """
    Tethys App Class for My First App.
    """

    name = 'My First App'
    index = 'my_first_app:home'
    icon = 'my_first_app/images/icon.gif'
    package = 'my_first_app'
    root_url = 'my-first-app'
    color = '#3498db'

    def url_maps(self):
        """
        Add controllers
        """
        UrlMap = url_map_maker(self.root_url)

        url_maps = (UrlMap(name='home',
                           url='my-first-app',
                           controller='my_first_app.controllers.home'),
                   UrlMap(name='map',
                           url='my-first-app/map',
                           controller='my_first_app.controllers.map'),
                   UrlMap(name='map_single',
                           url='my-first-app/map/{id}',
```

```

        controller='my_first_app.controllers.map_single'),
    UrlMap(name='echo_name',
           url='my-first-app/echo-name',
           controller='my_first_app.controllers.echo_name'),
)

return url_maps

def persistent_stores(self):
    """
    Add one or more persistent stores
    """
    stores = (PersistentStore(name='stream_gage_db',
                              initializer='init_stores:init_stream_gage_db',
                              spatial=True
                              ),
             )

    return stores

```

New Template

Create a new template called “echo_name.html” in your templates directory (my_first_app/templates/my_first_app/echo_name.html). Open the file and add the following contents:

```

{% extends "my_first_app/base.html" %}

{% load tethys_gizmos %}

{% block app_navigation_items %}
    <li class="active"><a href="{% url 'my_first_app:echo_name' %}">Name Echoer</a></li>
{% endblock %}

{% block app_content %}
    <form method="post">
        {% csrf_token %}
        {% gizmo text_input text_input_options %}
        <input type="submit" name="name-form-submit" class="btn btn-default">
    </form>

    {% if name %}
        <h1>Hello, {{ name }}!</h1>
    {% endif %}
{% endblock %}

{% block app_actions %}
    <a href="{% url 'my_first_app:home' %}" class="btn btn-default">Back</a>
{% endblock %}

```

The form is denoted by the HTML `<form>` tag and it contains a text input (created by a template Gizmo) and a submit button. Also note the use of the `csrf_token` tag. This is a security precaution that is required to be included in all the forms of your app (see the [Cross Site Forgery protection](#) article in the Django documentation for more details).

Also note that the method attribute of the `<form>` element is set to `post`. This means the form will use the HTTP method called POST to submit the data to the server. For an introduction to HTTP methods, see [The Definitive Guide to GET vs POST](#).

New Controller

Now you need to create the `echo_name` controller function. First, add the following import statement to the top of `my_first_app/controllers.py` file:

```
from tethys_gizmos.gizmo_options import TextInput
```

Then add the following function to your `my_first_app/controllers.py` file:

```
def echo_name(request):
    """
    Controller that will echo the name provided by the user via a form.
    """
    # Default value for name
    name = ''

    # Define Gizmo Options
    text_input_options = TextInput(display_text='Enter Name',
                                   name='name-input')

    # Check form data
    if request.POST and 'name-input' in request.POST:
        name = request.POST['name-input']

    # Create template context dictionary
    context = {'name': name,
              'text_input_options': text_input_options}

    return render(request, 'my_first_app/echo_name.html', context)
```

There are a few features to point out in this controller. First, the Gizmo options for the text input are defined in this controller via the `text_input_options` dictionary. The text input must have a name assigned to it for its value to be sent with the form data. In this case the name of the text input is “name-input”. See the [Template Gizmos API](#).

Next, the data that is submitted with HTML forms is returned through the `request` object. For forms submitted using the “post” method, the data will be accessible in the `request.POST` attribute. Similarly, form data submitted using the “get” method will be available via the `request.GET` attribute. Both `request.GET` and `request.POST` are dictionary like objects where the keys are the names of the fields from the form.

The controller contains logic that checks the `request.POST` for data with the name of the text input field, “name-input”. If it exists (which it will after the user submits the form), the `name` variable is replaced with the value of the text input. The `name` variable is passed to template where it renders a nice greeting.

Link to New Page

Create a link to the new page from the home page using the `url` tag. Open the `my_first_app/templates/my_first_app/home.html` file and replace the contents with this:

```
{% extends "my_first_app/base.html" %}

{% block app_navigation_items %}
    <li><a href="{% url 'my_first_app:echo_name' %}">Name Echoer</a></li>
{% endblock %}

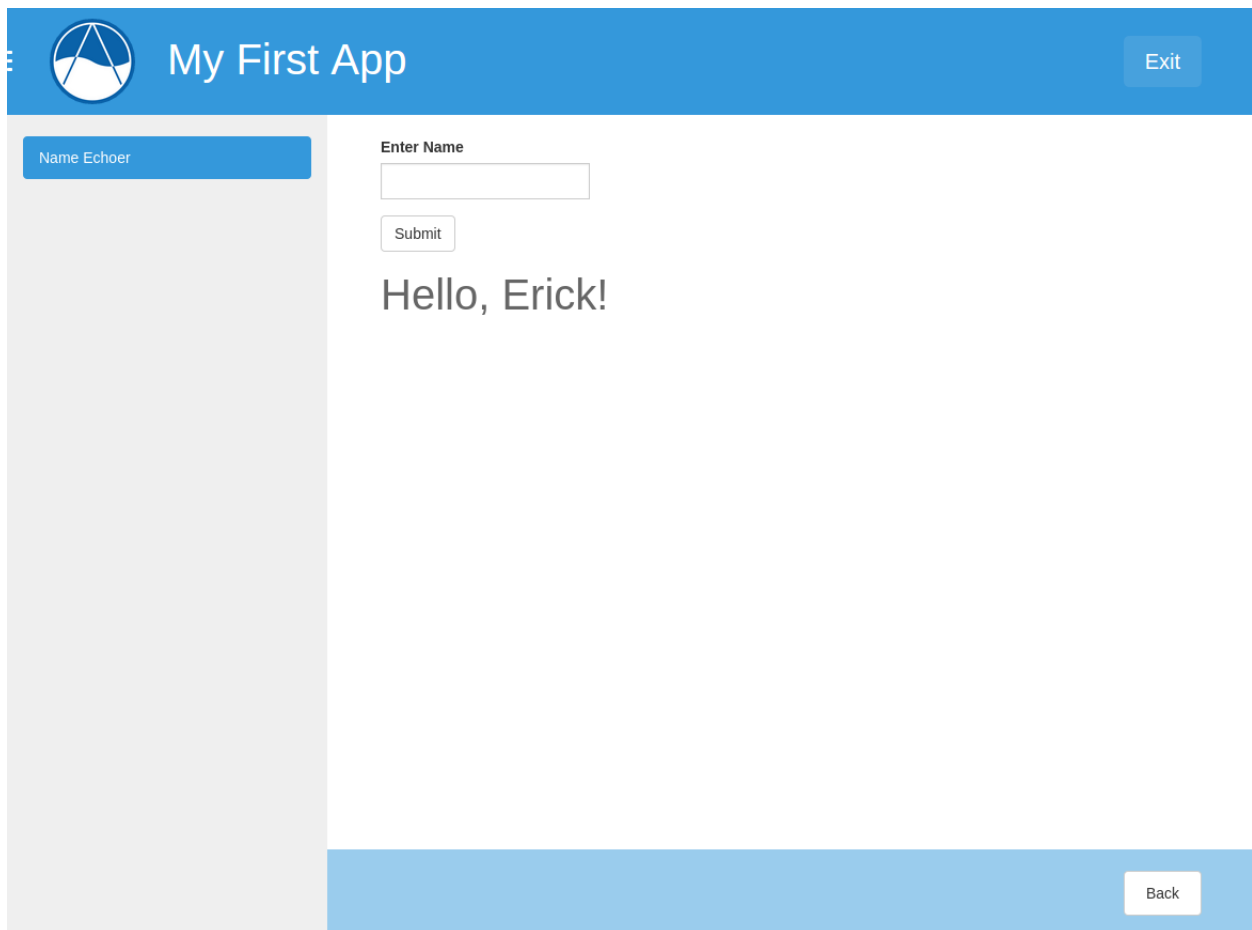
{% block app_content %}
    <h1>Welcome!</h1>
    <p>Congratulations on creating a new Tethys app!</p>
```

```
{% endblock %}

{% block app_actions %}
  <a href="{% url 'my_first_app:map' %}" class="btn btn-default">Go To Map</a>
{% endblock %}
```

View New Page

The app is ready to be tested. Run the **tethys manage start** command in the terminal and browse to your app. Use the “Name Echoer” link in the navigation to access the new page. Enter your name, press submit, and enjoy the greeting. Your new page should look something like this:



Distributing Apps

Last Updated: November 17, 2014

Once your app is complete, you will likely want to distribute it for others to use or at the very least install it in a production Tethys Platform environment. When you share your app with others, you will share the entire *release package*, which is the outermost directory of your *app project*. For these tutorials, your release package is called “tethysapp-my_first_app”.

The release package contains the source code for your app and a *setup script* (`setup.py`). You may also wish to include a README file and a LICENSE file in this directory. The *setup script* can be used to streamline installation of your app and any Python dependencies it may have. You already used the *setup script* without realizing it in the *Create a New Tethys App Project* tutorial when you installed your app for the first time (this command: `python setup.py develop`). A brief introduction to the *setup script* will be provided in this tutorial.

Setup Script

When you generate your app using the scaffold, it will automatically generate a *setup script* (`setup.py`). Open the *setup script* for your app located at `~/tethysdev/tethysapp-my_first_app/setup.py`. It should look something like this:

```
import os
import sys
from setuptools import setup, find_packages
from tethys_apps.app_installation import custom_develop_command, custom_install_command

### Apps Definition ###
app_package = 'my_first_app'
release_package = 'tethysapp-' + app_package
app_class = 'my_first_app.app:MyFirstApp'
app_package_dir = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'tethysapp',
                               app_package)
### Python Dependencies ###
dependencies = []

setup(
    name=release_package,
    version='0.0',
    description='',
    long_description='',
    keywords='',
    author='',
    author_email='',
    url='',
    license='',
    packages=find_packages(exclude=['ez_setup', 'examples', 'tests']),
    namespace_packages=['tethysapp', 'tethysapp.' + app_package],
    include_package_data=True,
    zip_safe=False,
    install_requires=dependencies,
    cmdclass={
        'install': custom_install_command(app_package, app_package_dir, dependencies),
        'develop': custom_develop_command(app_package, app_package_dir, dependencies)
    }
)
```

As a general rule, you should never modify the parameters under the “Apps Definition” heading. These parameters are used by the *setup script* to find the source code for your app and changing their values could result in your app not working properly. If you use Python libraries that are external to your app or Tethys Platform, you will need add the library name to the `dependencies` list in the *setup script*. These libraries will automatically be installed when your app is installed.

The final part of the setup script makes a call to the `setup()` function that is provided by the `setuptools` library. You will see the metadata that you defined during the scaffold process listed here. As you release subsequent versions of your app, you may wish to increment the `version` parameter of this function.

Setup Script Installation

The setup script is used to install your app and there are two types of installation that can be performed: `install` and `develop`. The `install` type of installation hard copies the source code of your app into the `site-packages` directory of your Python installation. The `site-packages` directory is where Python keeps all of the code for external modules and libraries that have been installed.

This is the type of installation you would use for a completed app that is being installed in a production environment. To perform this type of installation, open a terminal, change into the *release package* directory of your app, and run the `install` command on the *setup script* as follows:

```
cd ~/tethysdev/tethysapp-my_first_app
python setup.py install
```

The `install` type of installation is not well suited for working with your app during development, because you would need to reinstall it (i.e.: run the commands above) every time you made a change to the app source code. This is why the `develop` type of installation exists. When an app is installed with the `develop` command, the source code for your app is only linked to the `site-packages` directory. This allows you to change your code and test the changes without reinstalling the app.

You already performed this type of installation on your app during the *Create a New Tethys App Project* tutorial. To perform this type of installation, open a terminal, change into the *release package* directory, and run the `develop` command on the *setup script* like so:

```
cd ~/tethysdev/tethysapp-my_first_app
python setup.py develop
```

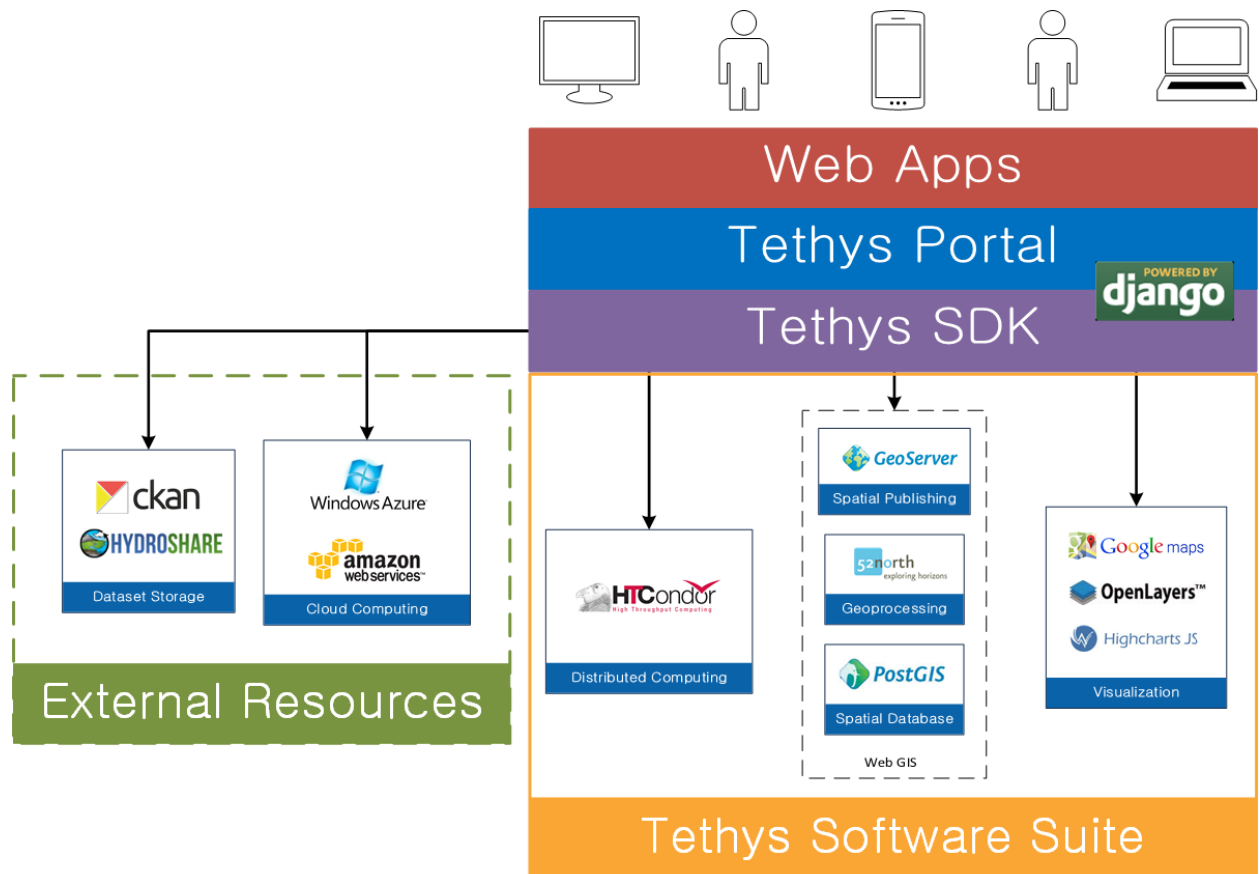
Tip: For more information about `setuptools` and the *setup script*, see the [Setuptools Documentation](#).

1.5 Software Suite

Last Updated: May 19, 2015

The Software Suite is the component of Tethys Platform that provides access to resources and functionality that are commonly required to develop water resources web apps. The primary motivation of creating the Tethys Software Suite was overcome the hurdle associated with selecting a GIS software stack to support spatial capabilities in apps. Some of the more specialized needs for water resources app development arise from the spatial data components of the models that are used in the apps. Distributed hydrologic models, for example, are parameterized using raster or vector layers such as land use maps, digital elevation models, and rainfall intensity grids.

A majority of the software projects included in the software suite are web GIS projects that can be used to acquire, modify, store, visualize, and analyze spatial data, but Tethys Software Suite also includes other software projects to address computing and visualization needs of water resources web apps. This article will describe the components included in the Tethys Software Suite in terms of the functionality provided by the software.



1.5.1 Spatial Database Storage



Tethys Software Suite includes the PostgreSQL database with PostGIS, a spatial database extension, to provide spatial data storage capabilities for Tethys web apps. PostGIS adds spatial column types including raster, geometry, and geography. The extension also provides database functions for basic analysis of GIS objects.

To use a PostgreSQL database in your app use the *Persistent Stores API*. To use a spatially enabled database with PostGIS use the *Spatial Persistent Stores API*.

1.5.2 Map Publishing



Tethys Software Suite provides GeoServer for publishing spatial data as web services. GeoServer is used to publish common spatial files such as Shapefiles and GeoTIFFs in web-friendly formats like GeoJSON and images.

To use the map publishing capabilities of GeoServer in your app use the *Spatial Dataset Services API*.

1.5.3 Geoprocessing

52°North Web Processing Service (WPS) is included in Tethys Software Suite as one means for supporting geoprocessing needs in water resources web app development. It can be linked with geoprocessing libraries such as GRASS, Sextante, and ArcGIS® Server for out-of-the-box geoprocessing capabilities.

The PostGIS extension, included in the software suite, can also provide geoprocessing capabilities on data that is stored in a spatially-enabled database. PostGIS includes SQL geoprocessing functions for splicing, dicing, morphing, reclassifying, and collecting/unioning raster and vector types. It also includes functions for vectorizing rasters, clipping rasters with vectors, and running stats on rasters by geometric region.

To use 52°North WPS or other WPS geoprocessing services in your app use the *Web Processing Services API*.

1.5.4 Visualization



OpenLayers 3 is a JavaScript web-mapping client library for rendering interactive maps on a web page. It is capable of displaying 2D maps of OGC web services and a myriad of other spatial formats and sources including GeoJSON, KML, GML, TopoJSON, ArcGIS REST, and XYZ.

To use an OpenLayers map in your app use the **Map View Gizmo** of the *Template Gizmos API*.



Google Maps™ provides the ability to render spatial data in a 2D mapping environment similar to OpenLayers, but it only supports displaying data in KML formats and data that are added via JavaScript API. Both maps provide a mechanism for drawing on the map for user input.

To use an OpenLayers map in your app use the **Google Map View Gizmo** of the *Template Gizmos API*.



Plotting capabilities are provided by [Highcharts](#), a JavaScript library created by Highsoft AS. The plots created using Highcharts are interactive with hovering effects, pan and zoom capabilities, and the ability to export the plots as images.

To use an OpenLayers map in your app use the **Plot View Gizmo** of the *Template Gizmos API*.

1.5.5 Distributed Computing



To facilitate the large-scale computing that is often required by water resources applications, Tethys Software Suite leverages the computing management middleware [HTCCondor](#). HTCCondor is both a resource management and a job scheduling software.

To use the HTCCondor and the computing capabilities in your app use the *Distributed Computing API*.

1.5.6 File Dataset Storage

Tethys Software Suite does not include software for handling flat file storage. However, Tethys SDK provides APIs for working with CKAN and HydroShare to address flat file storage needs. Descriptions of CKAN and HydroShare are provided here for convenience.



CKAN is an open source data sharing platform that streamlines publishing, sharing, finding, and using data. There is no central CKAN hub or portal, rather data publishers setup their own instance of CKAN to host the data for their organization.



HydroShare is an online hydrologic model and data sharing portal being developed by CUAHSI. It builds on the sharing capabilities of CUAHSI's Hydrologic Information System by adding support for sharing models and using social media functionality.

To use a CKAN instance for flat file storage in your app use the *Dataset Services API*. HydroShare is not fully supported at this time, but when it is you will use the *Dataset Services API* to access HydroShare resources.

1.5.7 Docker Installation



Tethys Software Suite uses [Docker](#) virtual container system to simplify the installation of some elements. Docker images are created and used to create containers, which are essentially stripped down virtual machines running only the software included in the image. Unlike virtual machines, the Docker containers do not partition the resources of your computer (processors, RAM, storage), but instead run as processes with full access to the resources of the computer.

Three Docker images are provided as part of Tethys Software Suite including:

- PostgreSQL with PostGIS
- 52° North WPS
- GeoServer.

The installation procedure for each software has been encapsulated in a Docker image reducing the installation procedure to three simple steps:

1. Install Docker
2. Download the Docker images
3. Deploy the Docker images as containers

1.5.8 SDK Relationships

Tethys Platform provides a software development kit (SDK) that provides application programming interfaces (APIs) for interacting with each of the software included in the Software Suite. The appropriate APIs are referenced in each section above, but a summary table of the relationship between the Software Suite and the SDK is provided as a reference.

Software	API	Functionality
PostgreSQL PostGIS	<i>Persistent Stores API</i> <i>Spatial Persistent Stores API</i>	SQL Database Storage Spatial Database Storage and Geoprocessing
GeoServer	<i>Spatial Dataset Services API</i>	Spatial File Publishing
52° North WPS	<i>Web Processing Services API</i>	Geoprocessing
OpenLayers, Google Maps, HighCharts	<i>Template Gizmos API</i>	Spatial and Tabular Visualization
HTCondor CKAN, HydroShare	<i>Distributed Computing API</i> <i>Dataset Services API</i>	Computing and Job Management Flat File Storage

1.6 Software Development Kit

Last Updated: May 13, 2015

The Tethys Platform provides a Python Software Development Kit (SDK) to make it easier to incorporate the functionality of the various supporting software packages into apps. The SDK includes an Application Programming Interface (API) for each of the major software components of Tethys Platform. This section contains the documentation for each API that is included in the SDK:

1.6.1 App Base Class API

Last Updated: November 20, 2014

Tethys apps are configured via the *app class*, which is contained in the *app configuration file* (`app.py`) of the *app project*. The *app class* must inherit from the `TethysAppBase` class to be loaded properly into CKAN. The following article contains the API documentation for the `TethysAppBase` class.

1.6.2 App Templating API

Last Updated: November 24, 2014

The pages of a Tethys app are created using the Django template language. This provides an overview of important Django templating concepts and introduces the base templates that are provided to make templating easier.

Django Templating Concepts

The Django template language allows you to create dynamic HTML templates and minimizes the amount of HTML you need to write for your app pages. This section will provide a crash course in Django template language basics, but we highly recommend a review of the [Django Template Language](#) documentation.

Tip: Review the [Django Template Language](#) to get a better grasp on templating in Tethys.

Variables

In Django templates, variables are denoted by double curly brace syntax: `{{ variable }}`. The variable expression will be replaced by the value of the variable. Dot notation can be used access attributes of a variable: `{{ variable.attribute }}`.

Examples:

```
# Examples of Django template variable syntax
{{ variable }}

# Access items in a list or tuple using dot notation
{{ list.0 }}

# Access items in a dictionary using dot notation
{{ dict.key }}

# Access attributes of objects using dot notation
{{ object.attribute }}
```

Hint: See [Django template Variables](#) documentation for more information.

Filters

Variables can be modified by filters which look like this: `{{ variable|filter:argument }}`. Filters perform modifying functions on variable output such as formatting dates, formatting numbers, changing the letter case, and concatenating multiple variables.

Examples:

```
# The default filter can be used to print a default value when the variable is falsy
{{ variable|default:"nothing" }}

# The join filter can be used to join a list with a the separator given
{{ list|join:", " }}
```

Hint: Refer to the [Django Filter Reference](#) for a full list of the filters available.

Tags

Tags use curly brace percent sign syntax like this: `{% tag %}`. Tags perform many different functions including creating text, controlling flow, or loading external information to be used in the app. Some commonly used tags include `for`, `if`, `block`, and `extends`.

Examples:

```
# The if tag only prints its contents when the condition evaluates to True
{% if name %}
    <h1>Hello, {{ name }}!</h1>
{% else %}
    <h1>Welcome!</h1>
{% endif %}

# The for tag can be used to loop through iterables printing its contents on each iteration
```



```
<ul>
  {% for item in item_list %}
    <li>{{ item }}</li>
  {% endfor %}
</ul>
```

```
# The block tag is used to override the contents of the block of a parent template
{% block example %}
  <p>I just overrode the contents of the "example" block with this paragraph.</p>
{% endblock %}
```

Hint: See the [Django Tag Reference](#) for a complete list of tags that Django provides.

Template Inheritance

One of the advantages of using the Django template language is that it provides a method for child templates to extend parent templates, which can reduce the amount of HTML you need to write. Template inheritance is accomplished using two tags, `extends` and `block`. Parent templates provide `blocks` of content that can be overridden by child templates. Child templates can extend parent templates by using the `extends` tag. Calling the `block` tag of a parent template in a child template will override any content in that `block` tag with the content in the child template.

Hint: The [Django Template Inheritance](#) documentation provides an excellent example that illustrates how inheritance works.

Base Templates

There are two layers of templates provided for Tethys app development. The `app_base.html` template provides the HTML skeleton for all Tethys app templates, which includes the base HTML structural elements (e.g.: `<html>`, `<head>`, and `<body>` elements), the base style sheets and JavaScript libraries, and many blocks for customization. All Tethys app projects also include a `base.html` template that inherits from the `app_base.html` template.

App developers are encouraged to use the `base.html` file as the base template for all of their templates, rather than extending the `app_base.html` file directly. The `base.html` template is easier to work with, because it includes only the blocks that will be used most often from the `app_base.html` template. However, all of the blocks that are available from `app_base.html` template will also be available for use in the `base.html` template and any templates that extend it.

Many of the blocks in the template correspond with different portions of the app interface. Figure 1 provides a graphical explanation of these blocks. An explanation of all the blocks provided in the `app_base.html` and `base.html` templates can be found in the section that follows.

Blocks

This section provides an explanation of the blocks are available for use in child templates of either the `app_base.html` or the `base.html` templates.

htmltag

Override the `<html>` element open tag.

Example:

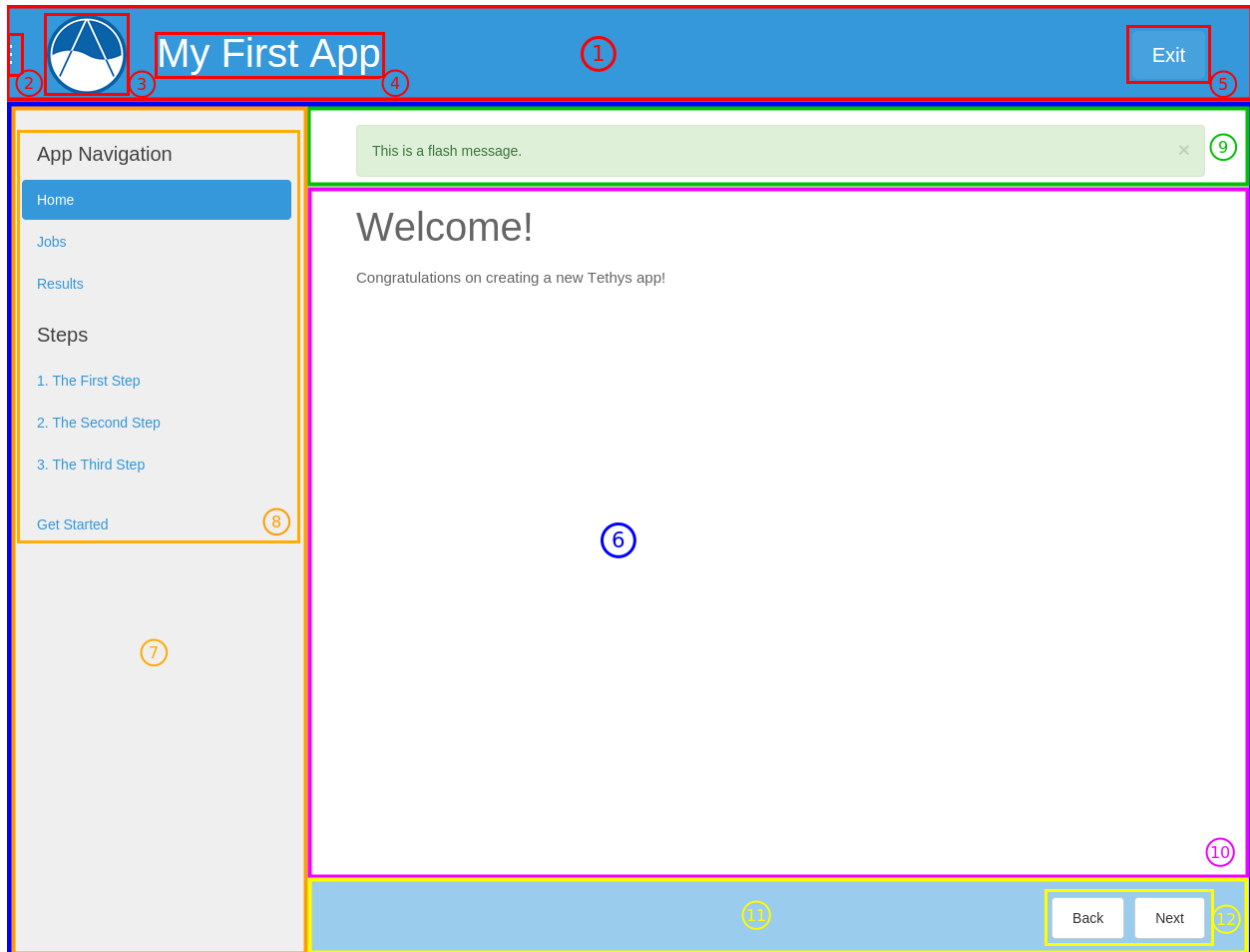


Figure 1.10: **Figure 1.** Illustration of the blocks that correspond with app interface elements as follows:

1. app_header_override
2. app_navigation_toggle_override
3. app_icon_override, app_icon
4. app_title_override, app_title
5. exit_button_override
6. app_content_override
7. app_navigation_override
8. app_navigation, app_navigation_items
9. flash
10. app_content
11. app_actions_override
12. app_actions

```
{% block htmltag %}<html lang="es">{% endblock %}
```

headtag

Add attributes to the `<head>` element.

Example:

```
{% block headtag %}style="display: block;"{% endblock %}
```

meta

Override or append `<meta>` elements to the `<head>` element. To append to existing elements, use `block.super`.

Example:

```
{% block meta %}
  {{ block.super }}
  <meta name="description" value="My website description" />
{% endblock %}
```

title

Change title for the page. The title is used as metadata for the site and shows up in the browser in tabs and bookmark names.

Example:

```
{% block title %}{{ block.super }} - My Sub Title{% endblock %}
```

links

Add content before the stylesheets such as rss feeds and favicons. Use `block.super` to preserve the default favicon or override completely to specify custom favicon.

Example:

```
{% block links %}
  <link rel="shortcut icon" href="/path/to/favicon.ico" />
{% endblock %}
```

styles

Add additional stylesheets to the page. Use `block.super` to preserve the existing styles for the app (recommended) or override completely to use your own custom stylesheets.

Example:

```
{% block styles %}
  {{ block.super }}
  <link href="/path/to/styles.css" rel="stylesheet" />
{% endblock %}
```

global_scripts

Add JavaScript libraries that need to be loaded prior to the page being loaded. This is a good block to use for libraries that are referenced globally. The global libraries included as global scripts by default are JQuery and Bootstrap. Use `block.super` to preserve the default global libraries.

Example:

```
{% block global_scripts %}
  {{ block.super }}
  <script src="/path/to/script.js" type="text/javascript"></script>
{% endblock %}
```

bodytag

Add attributes to the body element.

Example:

```
{% block bodytag %}class="a-class" onload="run_this();" {% endblock %}
```

app_content_wrapper_override

Override the app content structure completely. The app content wrapper contains all content in the `<body>` element other than the scripts. Use this block to override all of the app template structure completely.

Override Eliminates:

`app_header_override`, `app_navigation_toggle_override`, `app_icon_override`, `app_icon`, `app_title_override`, `app_title`, `exit_button_override`, `app_content_override`, `flash`, `app_navigation_override`, `app_navigation`, `app_navigation_items`, `app_content`, `app_actions_override`, `app_actions`.

Example:

```
{% block app_content_wrapper_override %}
  <div>
    <p>My custom content</p>
  </div>
{% endblock %}
```

app_header_override

Override the app header completely including any wrapping elements. Useful for creating a custom header for your app.

Override Eliminates:

`app_navigation_toggle_override`, `app_icon_override`, `app_icon`, `app_title_override`, `app_title`, `exit_button_override`

app_navigation_toggle_override

Override the app navigation toggle button. This is useful if you want to create an app that does not include the navigation pane. Use this to remove the navigation toggle button as well.

Example:

```
{% block app_navigation_toggle_override %}{% endblock %}
```

app_icon_override

Override the app icon in the header completely including any wrapping elements.

Override Eliminates:

app_icon

app_icon

Override the app icon `` element in the header.

Example:

```
{% block app_icon %}{% endblock %}
```

app_title_override

Override the app title in the header completely including any wrapping elements.

Override Eliminates:

app_title

app_title

Override the app title element in the header.

Example:

```
{% block app_title %}My App Title{% endblock %}
```

exit_button_override

Override the exit button completely including any wrapping elements.

app_content_override

Override only the app content area while preserving the header. The navigation and actions areas will also be overridden.

Override Eliminates:

flash, app_navigation_override, app_navigation, app_navigation_items, app_content, app_actions_override, app_actions

flash

Override the flash messaging capabilities. Flash messages are used to display dismissible messages to the user using the Django messaging capabilities. Override if you would like to implement your own messaging system or eliminate functionality all together.

app_navigation_override

Override the app navigation elements including any wrapping elements.

Override Eliminates:

app_navigation, app_navigation_items

app_navigation

Override the app navigation container. The default container for navigation is an unordered list. Use this block to override the unordered list for custom navigation.

Override Eliminates:

app_navigation_items

app_navigation_items

Override or append to the app navigation list. These should be `` elements.

app_content

Add content to the app content area. This should be the primary block used to add content to the app.

Example:

```
{% block app_content %}
  <p>Content for my app.</p>
{% endblock %}
```

app_actions_override

Override app content elements including any wrapping elements.

app_actions

Override or append actions to the action area. These are typically buttons or links. The actions are floated right, so they need to be listed in right to left order.

Example:

```
{% block app_actions %}
  <a href="" class="btn btn-default">Next</a>
  <a href="" class="btn btn-default">Back</a>
{% endblock %}
```

scripts

Add additional JavaScripts to the page. Use `block.super` to preserve the existing scripts for the app (recommended) or override completely to use your own custom scripts.

Example:

```
{% block scripts %}
  {{ block.super }}
  <script href="/path/to/script.js" type="text/javascript"></script>
{% endblock %}
```

app_base.html

This section provides the complete contents of the `app_base.html` template. It is meant to be used as a reference for app developers, so they can be aware of the HTML structure underlying their app templates.

```
{% load staticfiles tethys_gizmos %}
<!DOCTYPE html>

{% block htmltag %}
<!--[if IE 7]> <html lang="en" class="ie ie7"> <![endif]-->
<!--[if IE 8]> <html lang="en" class="ie ie8"> <![endif]-->
<!--[if IE 9]> <html lang="en" class="ie9"> <![endif]-->
<!--[if gt IE 8]><!--> <html lang="en" > <!--<![endif]-->
{% endblock %}

<head {% block headtag %}{% endblock %}>

  {% block meta %}
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="generator" content="Django" />
  {% endblock %}

  <title>
    {% if site_globals.site_title %}
      {{ site_globals.site_title }}
    {% elif site_globals.brand_text %}
      {{ site_globals.brand_text }}
    {% else %}
      Tethys
    {% endif %}
  {% block title %}{% endblock %}
</title>

  {% block links %}
    {% if site_globals.favicon %}
      <link rel="shortcut icon" href="{{ site_globals.favicon }}" />
    {% endif %}
  {% endblock %}

  {% block styles %}
    <link href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css"
      rel="stylesheet" /> <link href="{% static 'tethys_apps/css/app_base.css' %}"
      rel="stylesheet" />
  {% endblock %}
```

```

{% block global_scripts %}
  <script src="//code.jquery.com/jquery-2.1.1.min.js" type="text/javascript">
  </script>
  <script src="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/js/bootstrap.min.js"
  type="text/javascript"></script> {% endblock %}
</head>

<body {% block bodytag %}{% endblock %}>

{% block app_content_wrapper_override %}
  <div id="app-content-wrapper" class="show-nav">

    {% block app_header_override %}
      <div id="app-header" class="clearfix">
        <div class="tethys-app-header" style="background:
        {{ tethys_app.color|default:'#1b95dc' }};">
          {% block app_navigation_toggle_override %}
            <a href="javascript:void(0);" class="toggle-nav">
              <div></div>
              <div></div>
              <div></div>
            </a>
          {% endblock %}

          {% block app_icon_override %}
            <div class="icon-wrapper">
              {% block app_icon %}
              {% endblock %} </div>
          {% endblock %}

          {% block app_title_override %}
            <div class="app-title-wrapper">
              <span class="app-title">{% block app_title %}{{ tethys_app.name }}
              {% endblock %}</span></div>
          {% endblock %}

          {% block exit_button_override %}
            <div class="exit-button">
              <a href="javascript:void(0);" onclick="TETHYS_APP_BASE.exit_app('
              {% url 'app_library' %}</div>
            {% endblock %}
          </div>
        </div>
      </div>
    {% endblock %}

    {% block app_content_override %}
      <div id="app-content">

        {% block flash %}
          {% if messages %}
            <div class="flash-messages">
              {% for message in messages %}
                <div class="alert {% if message.tags %}{{ message.tags }}
                {% endif %}
                alert-dismissible"
                  <button type="button" class="close" data-dismiss="alert">
                    <span aria-hidden="true">&times;</span>
                    <span class="sr-only">Close</span>
                  </button>

```



```
        {{ message }}
      </div>
    {% endfor %}
  </div>
{% endif %}
{% endblock %}

{% block app_navigation_override %}
  <div id="app-navigation">
    {% block app_navigation %}
      <ul class="nav nav-pills nav-stacked">
        {% block app_navigation_items %}{% endblock %}
      </ul>
    {% endblock %}
  </div>
{% endblock %}

<div id="inner-app-content">
  {% block app_content %}{% endblock %}

  {# App actions are fixed to the bottom #}
  {% block app_actions_override %}
    <div id="app-actions">
      {% block app_actions %}{% endblock %}
    </div>
  {% endblock %}
</div>
</div>
{% endblock %}
</div>
{% endblock %}

{% block scripts %}
  <script src="{% static 'tethys_apps/vendor/cookies.js' %}" type="text/javascript">
</script><script src="{% static 'tethys_apps/js/app_base.js' %}"
type="text/javascript"></script> {% gizmo_dependencies %}

{% endblock %}
</body>
</html>
```

base.html

The `base.html` is the base template that is used directly by app templates. This file is generated in all new Tethys app projects that are created using the scaffold. The contents are provided here for reference.

All of the blocks provided by the `base.html` template are inherited from the `app_base.html` template. The `base.html` template is intended to be a simplified version of the `app_base.html` template, providing only the the blocks that should be used in a default app configuration. However, the blocks that are excluded from the `base.html` template can be used by advanced Tethys app developers who wish customize parts or all of the app template structure.

See the [Blocks](#) section for an explanation of each block.

```
{% extends "tethys_apps/app_base.html" %}

{% load staticfiles %}

{% block title %}- {{ tethys_app.name }}{% endblock %}
```

```

{% block styles %}
  {{ block.super }}
  <link href="{% static 'new_template_app/css/main.css' %}" rel="stylesheet"/>
{% endblock %}

{% block app_icon %}
  {# The path you provided in your app.py is accessible through the tethys_app.icon
  context variable 
{% endblock %}

{# The name you provided in your app.py is accessible through the tethys_app.name
context variable #}{% block app_title %}{{ tethys_app.name }}{% endblock %}

{% block app_navigation_items %}
  <li class="title">App Navigation</li>
  <li class="active"><a href="">Home</a></li>
  <li><a href="">Jobs</a></li>
  <li><a href="">Results</a></li>
  <li class="title">Steps</li>
  <li><a href="">1. The First Step</a></li>
  <li><a href="">2. The Second Step</a></li>
  <li><a href="">3. The Third Step</a></li>
  <li class="separator"></li>
  <li><a href="">Get Started</a></li>
{% endblock %}

{% block app_content %}
{% endblock %}

{% block app_actions %}
{% endblock %}

                                {% block scripts %}{{ block.super }}
  <script src="{% static 'new_template_app/js/main.js' %}" type="text/javascript"></
script> {% endblock %}

```

1.6.3 Template Gizmos API

Last Updated: May 21, 2015

Template Gizmos are building blocks that can be used to create beautiful interactive controls for web apps. Using the Template Gizmos API, developers can add date-pickers, plots, and maps to their app pages with minimal coding. This article provides an overview of how to use Gizmos. If you are not familiar with templating in Tethys apps, please review [The View and Templating](#) tutorial before proceeding.

For a detailed explanation and code examples of each Gizmo, see the [Gizmo Options Object API](#) documentation.

Working with Gizmos

The best way to illustrate how to use Template Gizmos is to look at an example. The following example illustrates how to add a date picker to a page using the Date Picker Gizmo. The basic workflow involves three steps:

1. Define gizmo options in the controller for the template
2. Load gizmo library in the template

3. Insert the gizmo tag in the template

A detailed description of each step follows.

1. Define Gizmo Options

The first step is to import the appropriate options object and configure the Gizmo. This is performed in the controller of the template where the Gizmo will be used.

In this case, we import `DatePickerOptions` and initialize a new object called `date_picker_options` with the appropriate options. Then we pass the object to the template via the `context` dictionary:

```
from tethys_gizmos.gizmos_options import DatePickerOptions

def example_controller(request):
    """
    Example of a controller that defines options for a Template Gizmo.
    """
    date_picker_options = DatePickerOptions(name='data1',
                                          display_text='Date',
                                          autoclose=True,
                                          format='MM d, yyyy',
                                          start_date='2/15/2014',
                                          start_view='decade',
                                          today_button=True,
                                          initial='February 15, 2014')

    context = {'date_picker_options': date_picker_options}

    return render(request, 'path/to/my/template.html', context)
```

Note: The Gizmo Options objects are new as of version 1.1.0. Prior to this time, Gizmo options were defined using dictionaries. The dictionary parameterization of Gizmos is still supported, but will no longer be referenced in the documentation.

The *Gizmo Options Object API* provides detailed descriptions of each Gizmo option object, valid parameters, and examples of how to use them.

2. Load Gizmo Library

Now near the top of the template where the Gizmo will be inserted, load the `tethys_gizmos` library using the Django load tag. This only needs to be done once per template:

```
{% load tethys_gizmos %}
```

3. Insert the Gizmo

The `gizmo` tag is used to insert the date picker anywhere in the template. The `gizmo` tag accepts two arguments: the name of the Gizmo to insert and a dictionary of configuration options for the Gizmo:

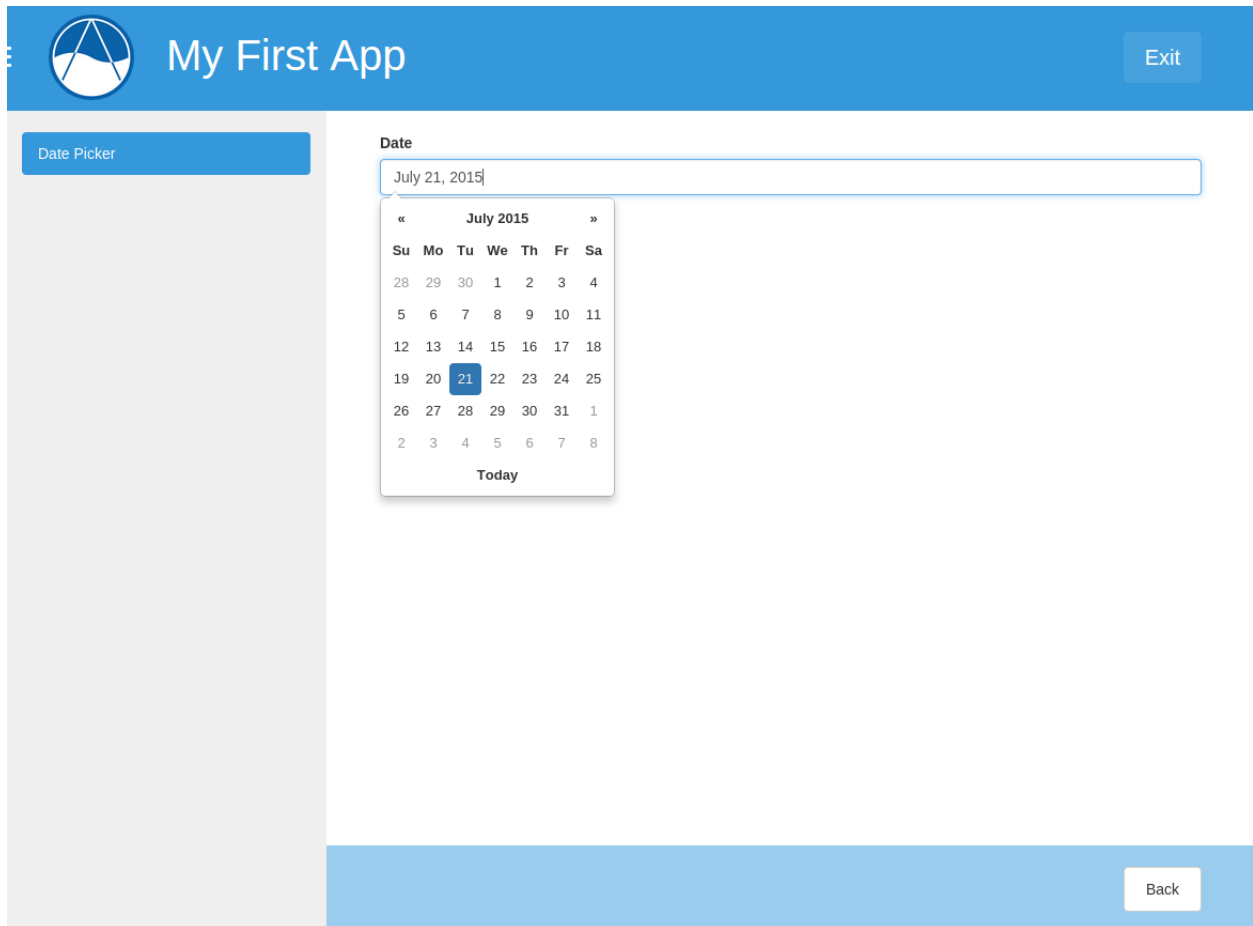
```
{% gizmo <name> <options> %}
```

For this example, the `date_picker` Gizmo is inserted and the `date_picker_options` object that was defined in the controller and passed to the template is provided:

```
{% gizmo date_picker date_picker_options %}
```

Rendered Gizmo

The Gizmo tag is replaced with the appropriate HTML, JavaScript, and CSS that is needed to render the Gizmo. In the example, the date picker is inserted at the location of the `gizmo` tag. The template with the rendered date picker would look something like this:



Gizmo Showcase

Live demos of each Gizmo is provided as a developer tool called “Gizmo Showcase”. To access the Gizmo Showcase, start up your development server and navigate to the home page of your Tethys Portal at <http://127.0.0.1:8000>. Login and select the `Developer` link from the main navigation. This will bring up the Developer Tools page of your Tethys Portal:

Select the Gizmos developer tool and you will be brought to the Gizmo Showcase page:

For explanations the Gizmo Options objects and code examples, refer to the *Gizmo Options Object API*.



Gizmos

Gizmos are building blocks that can be used to create beautiful interactive controls in Tethys Apps. Using gizmos, developers can add date-pickers, plots, and maps to their templates with minimal coding. Follow the link to learn more.

[Show me the docs.](#)



Dataset Services

Use this tool to browse the dataset services that are available for use in app development. Depending on what level of access you have to the dataset service, you may be able to view, update, create, and delete datasets.

[Go to tool.](#)



Geoprocessing

Geoprocessing in Tethys apps can be accomplished using the built-in 52 North WPS service. The Geoprocess Formulator tool can be used to view available 52 North processes and formulate the WPS request.

- [Quick Start](#)
- [Button Groups](#)
- [Date Picker](#)
- [Range Slider](#)
- [Select Input](#)
- [Text Input](#)
- [Toggle Switch](#)
- [Table View](#)
- [Plot View](#)
- [Message Box](#)
- [Google Map](#)
- [Map View](#)
- [Editable Google Map](#)
- [FetchClimate](#)

Gizmo Showcase

Gizmos are building blocks that can be used to create beautiful interactive controls for web apps. Using gizmos, developers can add date-pickers, plots, and maps to their templates with minimal coding. This page provides the documentation developers need to use Gizmos.

Quick Start

What does "minimal coding" mean? Take a look at the following example. Let's say you want to include a date picker in your template using a gizmo. First, create a dictionary with all the configuration options for the date picker (more on that later) in your view/controller for the template and add it to the context:

```
def my_view(request):
    date_picker_options = {'display_text': 'Date',
                          'name': 'date1',
                          'autoclose': True,
                          'format': 'MM d, yyyy',
                          'start_date': '2/15/2014',
                          'start_view': 'decade',
                          'today_button': True,
                          'initial': 'February 15, 2014'}

    context = {'date_picker_options': date_picker_options}

    return render(request, 'path/to/my/template.html', context)
```

Next, open the template you intend to add the gizmo to and load the **tethys_gizmos** library. Be sure to do this somewhere near the top of your template—before any gizmo occurrences. This only needs to be done once for each template that uses gizmos.

```
{% load tethys_gizmos %}
```

Django Tag Reference

This section contains a brief explanation of the template tags that power Gizmos. These are provided by the `tethys_gizmos` library that you load at the top of templates that use Gizmos.

gizmo

Inserts a Gizmo at the location of the tag.

Parameters:

- **name** (string or literal) - The name of the Gizmo to insert as either a string (e.g.: “date_picker”) or a literal (e.g.: `date_picker`).
- **options** (dict) - The configuration options for the Gizmo. The options are Gizmo specific. See the Gizmo Showcase documentation for descriptions of the options that are available.

Examples:

```
# With literal for name parameter
{% gizmo date_picker date_picker_options %}

# With string for name parameter
{% gizmo "date_picker" date_picker_options %}
```

gizmo_dependencies

Inserts the CSS and JavaScript dependencies at the location of the tag. This tag must appear after all occurrences of the `gizmo` tag. In Tethys Apps, these dependencies are imported for you, so this tag is not required. For external Django projects that use the `tethys_gizmos` Django app, this tag is required.

Parameters:

- **type** (string or literal, optional) - The type of dependency to import. This parameter can be used to include the CSS and JavaScript dependencies at different locations in the template. Valid values include “css” for CSS dependencies or “js” for JavaScript dependencies.

Examples:

```
# No type parameter
{% gizmo_dependencies %}

# CSS only
{% gizmo_dependencies css %}

# JavaScript only
{% gizmo_dependencies js %}
```

1.6.4 Gizmo Options Object API

Last Updated: May 25, 2015

This article provides explanations of each of the Gizmo Options Objects available for configuring Gizmos. It also provide code and usage examples for each object. For more explanation of Gizmos, see the [Template Gizmos API](#) documentation.

Button and Button Group

```
class tethys_gizmos.gizmo_options.Button (display_text='', name='', style='', icon='', href='',
                                          submit=False, disabled=False, attributes='',
                                          classes='')
```

display_text

str

Display text that appears on the button.

name

str

Name of the input element that will be used for form submission.

style

str

Name of the input element that will be used for form submission.

icon

str

Name of a valid Twitter Bootstrap icon class (see the Bootstrap [glyphicon](#) reference).

href

str

Link for anchor type buttons.

submit

bool

Set this to true to make the button a submit type button for forms.

disabled

bool

Set the disabled state.

attributes

str

A string representing additional HTML attributes to add to the primary element (e.g. “onclick=run_me();”).

classes

str

Additional classes to add to the primary HTML element (e.g. “example-class another-class”).

Example:

```
# CONTROLLER

from tethys_apps.sdk.gizmos import Button

# Single Button
single_button = Button(display_text='Click Me',
                       name='click_me_name',
                       attributes='onclick=alert(this.name);',
                       submit=True)
```



```
# TEMPLATE

{% gizmo button single_button %}
```

class `tethys_gizmos.gizmo_options.ButtonGroup` (*buttons*, *vertical=False*, *attributes=''*, *classes=''*)

The button group gizmo can be used to generate a single button or a group of buttons. Groups of buttons can be stacked horizontally or vertically. For a single button, specify a button group with one button. This gizmo is a wrapper for Twitter Bootstrap buttons.

buttons

list, required

A list of dictionaries where each dictionary contains the options for a button.

vertical

bool

Set to true to have button group stack vertically.

attributes

str

A string representing additional HTML attributes to add to the primary element (e.g. “onclick=run_me();”).

classes

str

Additional classes to add to the primary HTML element (e.g. “example-class another-class”).

Example

```
# CONTROLLER

from tethys_apps.sdk.gizmos import Button, ButtonGroup

# Horizontal Button Group
add_button = Button(display_text='Add',
                    icon='glyphicon glyphicon-plus',
                    style='success')
delete_button = Button(display_text='Delete',
                      icon='glyphicon glyphicon-trash',
                      disabled=True,
                      style='danger')
horizontal_buttons = ButtonGroup(buttons=[add_button, delete_button])

# Vertical Button Group
edit_button = Button(display_text='Edit',
                    icon='glyphicon glyphicon-wrench',
                    style='warning',
                    attributes='id=edit_button')
info_button = Button(display_text='Info',
                    icon='glyphicon glyphicon-question-sign',
                    style='info',
                    attributes='name=info')
apps_button = Button(display_text='Apps',
                    icon='glyphicon glyphicon-home',
                    href='/apps',
                    style='primary')
vertical_buttons = ButtonGroup(buttons=[edit_button, info_button, apps_button],
                              vertical=True)
```

```
# TEMPLATE

{% gizmo button_group horizontal_buttons %}
{% gizmo button_group vertical_buttons %}
```

Date Picker

```
class tethys_gizmos.gizmo_options.DatePicker(name, display_text='', autoclose=False,
calendar_weeks=False, clear_button=False,
days_of_week_disabled='', end_date='',
format='', min_view_mode='days', multi-
date=1, start_date='', start_view='month',
today_button=False, today_highlight=False,
week_start=0, initial='', disabled=False,
error='', attributes='', classes='')
```

Date pickers are used to make the input of dates streamlined and easy. Rather than typing the date, the user is presented with a calendar to select the date. This date picker was implemented using [Bootstrap Datepicker](#).

name

str, required

Name of the input element that will be used for form submission.

display_text

str

Display text for the label that accompanies date picker.

autoclose

bool

Set whether datepicker auto closes when a date is selected.

calendar_weeks

bool

Set whether calendar week numbers are shown on the left of the datepicker.

clear_button

bool

Set whether the clear button is displayed or not.

days_of_week_disabled

str

Days of the week that are disabled 0-6 with 0 being Sunday and 6 being Saturday. Multiple days are comma separated (e.g.: '0,6').

end_date

str

Last date that can be selected. All other dates after this date are shown as disabled.

format

str

String representing date format. For valid formats see [Bootstrap Datepicker documentation here](#).

min_view_mode

str

Set the minimum view mode. Possible values are 'days', 'months', 'years'.

multidate

int

Enables multi-selection of dates up to the number given.

start_date

str

First date that can be selected. All other dates before this date are shown as disabled.

start_view

str

View the date picker starts on. Valid values include 'month', 'year', and 'decade'.

today_button

bool

Set whether a today button is displayed or not.

today_highlight

bool

Set whether to highlight the current date.

week_start

int

Set the day the week starts on 0-6, where 0 is Sunday and 6 is Saturday.

initial

str

Initial date to appear in date picker.

disabled

bool

Disabled state of the date picker.

error

str

Error message for form validation.

attributes

str

A string representing additional HTML attributes to add to the primary element (e.g. "onclick=run_me();").

classes

str

Additional classes to add to the primary HTML element (e.g. "example-class another-class").

Example

```
# CONTROLLER

from tethys_apps.sdk.gizmos import DatePickerOptions

# Date Picker Options
date_picker = DatePicker(name='date1',
```

```

        display_text='Date',
        autoclose=True,
        format='MM d, YYYY',
        start_date='2/15/2014',
        start_view='decade',
        today_button=True,
        initial='February 15, 2014')

date_picker_error = DatePicker(name='data2',
                               display_text='Date',
                               initial='10/2/2013',
                               disabled=True,
                               error='Here is my error text.')

# TEMPLATE

{% gizmo date_picker date_picker %}
{% gizmo date_picker date_picker_error %}

```

Range Slider

class `tethys_gizmos.gizmo_options.RangeSlider` (*name, min, max, initial, step, disabled=False, display_text='', error='', attributes='', classes=''*)

Sliders can be used to request an input value from a range of possible values. A slider is configured with a dictionary of key-value options. The table below summarizes the options for sliders.

display_text

str

Display text for the label that accompanies slider

name

str, required

Name of the input element that will be used on form submission

min

int, required

Minimum value of range

max

int, required

Maximum value of range

initial

int, required

Initial value of slider

step

int, required

Increment between values in range

disabled

bool

Disabled state of the slider

error

str

Error message for form validation

attributes

str

A string representing additional HTML attributes to add to the primary element (e.g. “onclick=run_me();”).

classes

str

Additional classes to add to the primary HTML element (e.g. “example-class another-class”).

Example

```
# CONTROLLER
from tethys_apps.sdk.gizmos import RangeSlider

slider1 = RangeSlider(display_text='Slider 1',
                      name='slider1',
                      min=0,
                      max=100,
                      initial=50,
                      step=1)

slider2 = RangeSlider(display_text='Slider 2',
                      name='slider2',
                      min=0,
                      max=1,
                      initial=0.5,
                      step=0.1,
                      disabled=True,
                      error='Incorrect, please choose another value.')

# TEMPLATE

{% gizmo range_slider slider1 %}
{% gizmo range_slider slider2 %}
```

Select Input

class `tethys_gizmos.gizmo_options.SelectInput` (*name*, *display_text*='', *multiple*=False, *original*=False, *options*='', *disabled*=False, *error*='', *attributes*='', *classes*='')

Select inputs are used to select values from an given set of values. Use this gizmo to create select inputs and multi select inputs. This uses the Select2 functionality.

display_text

str

Display text for the label that accompanies select input

name

str, required

Name of the input element that will be used for form submission

multiple*bool*

If True, select input will be a multi-select

original*bool*If True, [Select2 reference](#) functionality will be turned off**options***list*

List of tuples that represent the options and values of the select input

disabled*bool*

Disabled state of the select input

error*str*

Error message for form validation

attributes*str*

A string representing additional HTML attributes to add to the primary element (e.g. “onclick=run_me();”).

classes*str*

Additional classes to add to the primary HTML element (e.g. “example-class another-class”).

Example

```
# CONTROLLER
from tethys_apps.sdk.gizmos import SelectInput

select_input2 = SelectInput(display_text='Select2',
                            name='select1',
                            multiple=False,
                            options=[('One', '1'), ('Two', '2'), ('Three', '3')],
                            original=['Two'])

select_input2_multiple = SelectInput(display_text='Select2 Multiple',
                                     name='select2',
                                     multiple=True,
                                     options=[('One', '1'), ('Two', '2'),
                                              ('Three', '3')])

select_input_multiple = SelectInput(display_text='Select Multiple',
                                    name='select2.1',
                                    multiple=True,
                                    original=True,
                                    options=[('One', '1'), ('Two', '2'),
                                             ('Three', '3')])

select_input2_error = SelectInput(display_text='Select2 Disabled',
                                  name='select3',
                                  multiple=False,
                                  options=[('One', '1'), ('Two', '2'),
                                           ('Three', '3')], disabled=True,
```

```
error='Here is my error text')

# TEMPLATE

{% gizmo select_input select_input2 %}
{% gizmo select_input select_input2_multiple %}
{% gizmo select_input select_input_multiple %}
{% gizmo select_input select_input2_error %}
```

Text Input

```
class tethys_gizmos.gizmo_options.TextInput(name, display_text='', initial='', place-
                                             holder='', prepend='', append='',
                                             icon_prepend='', icon_append='', dis-
                                             abled=False, error='', attributes='',
                                             classes='')
```

The text input gizmo makes it easy to add text inputs to your app that are styled similarly to the other input snippets.

display_text

str

Display text for the label that accompanies select input

name

str, required

Name of the input element that will be used for form submission

initial

str

The initial text that will appear in the text input when it loads

placeholder

str

Placeholder text is static text that displayed in the input when it is empty

prepend

str

Text that is prepended to the text input

append

str

Text that is appended to the text input

icon_prepend

str

The name of a valid Bootstrap v2.3 icon. The icon will be prepended to the input.

icon_append

str

The name of a valid Bootstrap v2.3 icon. The icon will be appended to the input.

disabled

bool

Disabled state of the select input

error

str

Error message for form validation

attributes

str

A string representing additional HTML attributes to add to the primary element (e.g. “onclick=run_me();”).

classes

str

Additional classes to add to the primary HTML element (e.g. “example-class another-class”).

Example

```
# CONTROLLER
from tethys_apps.sdk.gizmos import TextInput

text_input = TextInput(display_text='Text',
                      name='inputAmount',
                      placeholder='e.g.: 10.00',
                      prepend='$')

text_error_input = TextInput(display_text='Text Error',
                             name='inputEmail',
                             initial='bob@example.com',
                             disabled=True,
                             icon_append='glyphicon glyphicon-envelope',
                             error='Here is my error text')

# TEMPLATE

{% gizmo text_input text_input %}
{% gizmo text_input text_error_input %}
```

Toggle Switch

```
class tethys_gizmos.gizmo_options.ToggleSwitch(name, display_text='', on_label='ON',
                                              off_label='OFF', on_style='primary',
                                              off_style='default', size='regular', initial=False,
                                              disabled=False, error='',
                                              attributes='', classes='')
```

Toggle switches can be used as an alternative to check boxes for boolean or binomial input. Toggle switches are implemented using the excellent [Bootstrap Switch](#) reference project.

display_text

str

Display text for the label that accompanies switch

name

str, required

Name of the input element that will be used for form submission

on_label

str

Text that appears in the “on” position of the switch

off_label

str

Text that appears in the “off” position of the switch

on_style

str

Color of the “on” position. Either: ‘default’, ‘info’, ‘primary’, ‘success’, ‘warning’, or ‘danger’

off_style

str

Color of the “off” position. Either: ‘default’, ‘info’, ‘primary’, ‘success’, ‘warning’, or ‘danger’

size

str

Size of the switch. Either: ‘large’, ‘small’, or ‘mini’.

initial

bool

The initial position of the switch (True for “on” and False for “off”)

disabled

bool

Disabled state of the switch

error

str

Error message for form validation

attributes

str

A string representing additional HTML attributes to add to the primary element (e.g. “onclick=run_me();”).

classes

str

Additional classes to add to the primary HTML element (e.g. “example-class another-class”).

Example

```
# CONTROLLER
from tethys_apps.sdk.gizmos import ToggleSwitch

toggle_switch = ToggleSwitch(display_text='Default Toggle',
                              name='toggle1')

toggle_switch_styled = ToggleSwitch(display_text='Styled Toggle',
                                     name='toggle2',
                                     on_label='Yes',
                                     off_label='No',
                                     on_style='success',
                                     off_style='danger',
```

```

        initial=True,
        size='large')

toggle_switch_disabled = ToggleSwitch(display_text='Disabled Toggle',
                                       name='toggle3',
                                       on_label='On',
                                       off_label='Off',
                                       on_style='success',
                                       off_style='warning',
                                       size='mini',
                                       initial=False,
                                       disabled=True,
                                       error='Here is my error text')

# TEMPLATE

{% gizmo toggle_switch toggle_switch %}
{% gizmo toggle_switch toggle_switch_styled %}
{% gizmo toggle_switch toggle_switch_disabled %}

```

Message Box

```

class tethys_gizmos.gizmo_options.MessageBox(name, title, message='', dismiss_button='Cancel', affirmative_button='Ok', affirmative_attributes='', width=560, attributes='', classes='')

```

Message box gizmos can be used to display messages to users. These are especially useful for alerts and warning messages. The message box gizmo is implemented using Twitter Bootstrap's modal.

name

str, required

Unique name for the message box

title

str, required

Title that appears at the top of the message box

message

str

Message that will appear in the main body of the message box

dismiss_button

str

Title for the dismiss button (a.k.a.: the "Cancel" button)

affirmative_button

str

Title for the affirmative action button (a.k.a.: the "OK" button)

affirmative_attributes

str

Use this to place any html attributes on the affirmative button. (e.g.: 'href="/action" onclick="doSomething();"')

width

int

The width of the message box in pixels

attributes

str

A string representing additional HTML attributes to add to the primary element (e.g. “onclick=run_me();”).

classes

str

Additional classes to add to the primary HTML element (e.g. “example-class another-class”).

Example

```
# CONTROLLER
from tethys_apps.sdk.gizmos import MessageBox

message_box = MessageBox(name='sampleModal',
                        title='Message Box Title',
                        message='Congratulations! This is a message box.',
                        dismiss_button='Nevermind',
                        affirmative_button='Proceed',
                        width=400,
                        affirmative_attributes='href=javascript:void(0);')

# TEMPLATE

{% gizmo message_box message_box %}
```

Table View

```
class tethys_gizmos.gizmo_options.TableView(rows, column_names='', hover=False,
                                             striped=False, bordered=False, condensed=False,
                                             editable_columns='', row_ids='', attributes='', classes='')
```

Table views can be used to display tabular data. The table view gizmo can be configured to have columns that are editable. When used in this capacity, embed the table view in a form with a submit button.

rows

tuple or list, required

A list/tuple of lists/tuples representing each row in the table.

column_names

tuple or list

A tuple or list of strings that represent the table columns names.

hover

bool

Illuminate rows on hover (does not work on striped tables)

striped

bool

Stripe rows

bordered*bool*

Add borders and rounded corners

condensed*bool*

A more tightly packed table

editable_columns*list or tuple*

A list or tuple with an entry for each column in the table. The entry is either `False` for non-editable columns or a string that will be used to create identifiers for the input fields in that column.

row_ids*list or tuple*

A list or tuple of ids for each row in the table. These will be combined with the string in the `editable_columns` parameter to create unique identifiers for easy input field in the table. If not specified, each row will be assigned an integer value.

attributes*str*

A string representing additional HTML attributes to add to the primary element (e.g. `"onclick=run_me();"`).

classes*str*

Additional classes to add to the primary HTML element (e.g. `"example-class another-class"`).

Example

```
# CONTROLLER
from tethys_apps.sdk.gizmos import TableView

table_view = TableView(column_names=('Name', 'Age', 'Job'),
                       rows=[('Bill', 30, 'contractor'),
                              ('Fred', 18, 'programmer'),
                              ('Bob', 26, 'boss')],
                       hover=True,
                       striped=False,
                       bordered=False,
                       condensed=False)

table_view_edit = TableView(column_names=('Name', 'Age', 'Job'),
                            rows=[('Bill', 30, 'contractor'),
                                   ('Fred', 18, 'programmer'),
                                   ('Bob', 26, 'boss')],
                            hover=True,
                            striped=True,
                            bordered=False,
                            condensed=False,
                            editable_columns=(False, 'ageInput', 'jobInput'),
                            row_ids=[21, 25, 31])

# TEMPLATE
```

```
{% gizmo table_view table_view %}
{% gizmo table_view table_view_edit %}
```

Plot View

class `tethys_gizmos.gizmo_options.PlotView` (*highcharts_object*, *height='520px'*,
width='100%', attributes='', classes='')

The Plot View gizmo can be used to generate interactive plots of tabular data. All of the plots available through this gizmo are powered by the [Highcharts](#) JavaScript library.

highcharts_object

HighChartsObject, required

The `highcharts_object` contains the definition of the plot. The full [Highcharts API reference](#) is supported via this object. The object can either be a JavaScript string, a JavaScript-equivalent Python data structure, or one of the supported options objects that are described below. The latter is recommended.

height

str

Height of the plot element. Any valid css unit of length.

width

str

Width of the plot element. Any valid css unit of length.

attributes

str

A string representing additional HTML attributes to add to the primary element (e.g. “onclick=run_me();”).

classes

str

Additional classes to add to the primary HTML element (e.g. “example-class another-class”).

Plots are configured using the `highcharts_object` parameter, which accepts either one of the Options objects or a dictionary equivalent of a HighCharts JavaScript object. The Options objects provide the easiest option for creating Plot Views (see the examples below).

Alternatively, developers can specify any valid option from the [Highcharts JavaScript API](#) using equivalent Python dictionaries. The recommended strategy for creating custom plots using dictionaries and the Highcharts Plot View is to find a similar example from the [Highcharts demos](#) page. Use the *Edit in jsFiddle* link to view the source code. Use the JavaScript source of the example to construct the equivalent in Python. Here are a few tips for converting from JavaScript to Python:

- The Python equivalent of JavaScript objects are dictionaries. However, the keys for Python dictionaries must be strings, not literals.
- Python datetime objects should be used for times (see the timeseries plot example).
- Some Highcharts API parameters require JavaScript functions to be passed in (e.g.: axis scale labels). To accomplish this in Python, pass a string representing the equivalent JavaScript function definition. These function definitions *must* be anonymous (no name). An example of this is shown in the line plot example.
- Use the Python syntax for Booleans (e.g.: True instead of true).

Note: To use non ASCII characters such as degree signs (°), you must declare utf-8 encoding at the top of your Python script: `# coding=utf-8`

Example

```

# CONTROLLER

# coding=utf-8
from tethys_apps.sdk.gizmos import *

### Line Plot

highcharts_object = HighChartsLinePlot(title='Plot Title',
                                       subtitle='Plot Subtitle',
                                       spline=True,
                                       x_axis_title='Altitude',
                                       x_axis_units='km',
                                       y_axis_title='Temperature',
                                       y_axis_units='°C',
                                       series=[
                                           {
                                               'name': 'Air Temp',
                                               'color': '#0066ff',
                                               'marker': {'enabled': False},
                                               'data': [
                                                   [0, 5], [10, -70],
                                                   [20, -86.5], [30, -66.5],
                                                   [40, -32.1],
                                                   [50, -12.5], [60, -47.7],
                                                   [70, -85.7], [80, -106.5]
                                               ]
                                           },
                                           {
                                               'name': 'Water Temp',
                                               'color': '#ff6600',
                                               'data': [
                                                   [0, 15], [10, -50],
                                                   [20, -56.5], [30, -46.5],
                                                   [40, -22.1],
                                                   [50, -2.5], [60, -27.7],
                                                   [70, -55.7], [80, -76.5]
                                               ]
                                           }
                                       ]
)

line_plot_view = PlotView(highcharts_object=highcharts_object,
                           width='500px',
                           height='500px')

### Scatter Plot

male_dataset = {
    'name': 'Male',
    'color': '#0066ff',
    'data': [
        [174.0, 65.6], [175.3, 71.8], [193.5, 80.7], [186.5, 72.6],
        [187.2, 78.8], [181.5, 74.8], [184.0, 86.4], [184.5, 78.4],
        [175.0, 62.0], [184.0, 81.6], [180.0, 76.6], [177.8, 83.6],
        [192.0, 90.0], [176.0, 74.6], [174.0, 71.0], [184.0, 79.6],
        [192.7, 93.8], [171.5, 70.0], [173.0, 72.4], [176.0, 85.9],
        [176.0, 78.8], [180.5, 77.8], [172.7, 66.2], [176.0, 86.4],
    ]
}

```

```

    [173.5, 81.8], [178.0, 89.6], [180.3, 82.8], [180.3, 76.4],
    [164.5, 63.2], [173.0, 60.9], [183.5, 74.8], [175.5, 70.0],
    [188.0, 72.4], [189.2, 84.1], [172.8, 69.1], [170.0, 59.5],
    [182.0, 67.2], [170.0, 61.3], [177.8, 68.6], [184.2, 80.1],
    [186.7, 87.8], [171.4, 84.7], [172.7, 73.4], [175.3, 72.1],
    [180.3, 82.6], [182.9, 88.7], [188.0, 84.1], [177.2, 94.1],
    [172.1, 74.9], [167.0, 59.1], [169.5, 75.6], [174.0, 86.2],
    [172.7, 75.3], [182.2, 87.1], [164.1, 55.2], [163.0, 57.0],
    [171.5, 61.4], [184.2, 76.8], [174.0, 86.8], [174.0, 72.2],
    [177.0, 71.6], [186.0, 84.8], [167.0, 68.2], [171.8, 66.1]
  ]
}

female_dataset = {
  'name': 'Female',
  'color': '#ff6600',
  'data': [
    [161.2, 51.6], [167.5, 59.0], [159.5, 49.2], [157.0, 63.0],
    [155.8, 53.6], [170.0, 59.0], [159.1, 47.6], [166.0, 69.8],
    [176.2, 66.8], [160.2, 75.2], [172.5, 55.2], [170.9, 54.2],
    [172.9, 62.5], [153.4, 42.0], [160.0, 50.0], [147.2, 49.8],
    [168.2, 49.2], [175.0, 73.2], [157.0, 47.8], [167.6, 68.8],
    [159.5, 50.6], [175.0, 82.5], [166.8, 57.2], [176.5, 87.8],
    [170.2, 72.8], [174.0, 54.5], [173.0, 59.8], [179.9, 67.3],
    [170.5, 67.8], [160.0, 47.0], [154.4, 46.2], [162.0, 55.0],
    [176.5, 83.0], [160.0, 54.4], [152.0, 45.8], [162.1, 53.6],
    [170.0, 73.2], [160.2, 52.1], [161.3, 67.9], [166.4, 56.6],
    [168.9, 62.3], [163.8, 58.5], [167.6, 54.5], [160.0, 50.2],
    [161.3, 60.3], [167.6, 58.3], [165.1, 56.2], [160.0, 50.2],
    [170.0, 72.9], [157.5, 59.8], [167.6, 61.0], [160.7, 69.1],
    [163.2, 55.9], [152.4, 46.5], [157.5, 54.3], [168.3, 54.8],
    [180.3, 60.7], [165.5, 60.0], [165.0, 62.0], [164.5, 60.3]
  ]
}

highcharts_object = HighChartsScatterPlot(title='Scatter Plot',
                                          subtitle='Scatter Plot',
                                          x_axis_title='Height',
                                          x_axis_units='cm',
                                          y_axis_title='Weight',
                                          y_axis_units='kg',
                                          series=[male_dataset, female_dataset]
)

scatter_plot_view = PlotView(highcharts_object=highcharts_object,
                              width='500px',
                              height='500px')

### Polar Plot

web_plot_object = HighChartsPolarPlot(title='Polar Chart',
                                       subtitle='Polar Chart',
                                       pane={
                                         'size': '80%'
                                       },
                                       categories=['Infiltration', 'Soil Moisture', 'Precipitation',
                                                  'Evaporation',
                                                  'Roughness', 'Runoff', 'Permeability', 'Vegetation']
)

```

```

        series=[
            {
                'name': 'Park City',
                'data': [0.2, 0.5, 0.1, 0.8, 0.2, 0.6, 0.8, 0.3],
                'pointPlacement': 'on'
            },
            {
                'name': 'Little Dell',
                'data': [0.8, 0.3, 0.2, 0.5, 0.1, 0.8, 0.2, 0.6],
                'pointPlacement': 'on'
            }
        ]
    )

web_plot = PlotView(highcharts_object=web_plot_object,
                    width='500px',
                    height='500px')

### Pie Plot

pie_plot_object = HighChartsPiePlot(title='Pie Chart',
                                    subtitle='Pie Chart',
                                    series=[{
                                        'type': 'pie',
                                        'name': 'Browser share',
                                        'data': [
                                            ['Firefox', 45.0],
                                            ['IE', 26.8],
                                            {
                                                'name': 'Chrome',
                                                'y': 12.8,
                                                'sliced': True,
                                                'selected': True
                                            },
                                            ['Safari', 8.5],
                                            ['Opera', 6.2],
                                            ['Others', 0.7]
                                        ]
                                    }
                                    ])

pie_plot_view = PlotView(highcharts_object=pie_plot_object,
                          width='500px',
                          height='500px')

### Bar Plot

bar_plot_view = HighChartsBarPlot(
    title='Bar Chart',
    subtitle='Bar Chart',
    vertical=False,
    categories=[
        'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
        'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],
    y_axis_units='millions',
    y_axis_title='Population',
    series=[{
        'name': 'Year 1800',

```



```
        'data': [107, 31, 635, 203, 2]
    }, {
        'name': 'Year 1900',
        'data': [133, 156, 947, 408, 6]
    }, {
        'name': 'Year 2008',
        'data': [973, 914, 4054, 732, 34]}
    ]
)

bar_plot_view = PlotView(highcharts_object=bar_plot_view,
                        width='500px',
                        height='500px')

### Time Series Plot

timeseries_plot_object = HighChartsTimeSeries(
    title='Irregular Timeseries Plot',
    y_axis_title='Snow depth',
    y_axis_units='m',
    series=[{
        'name': 'Winter 2007-2008',
        'data': [
            [datetime(2008, 12, 2), 0.8],
            [datetime(2008, 12, 9), 0.6],
            [datetime(2008, 12, 16), 0.6],
            [datetime(2008, 12, 28), 0.67],
            [datetime(2009, 1, 1), 0.81],
            [datetime(2009, 1, 8), 0.78],
            [datetime(2009, 1, 12), 0.98],
            [datetime(2009, 1, 27), 1.84],
            [datetime(2009, 2, 10), 1.80],
            [datetime(2009, 2, 18), 1.80],
            [datetime(2009, 2, 24), 1.92],
            [datetime(2009, 3, 4), 2.49],
            [datetime(2009, 3, 11), 2.79],
            [datetime(2009, 3, 15), 2.73],
            [datetime(2009, 3, 25), 2.61],
            [datetime(2009, 4, 2), 2.76],
            [datetime(2009, 4, 6), 2.82],
            [datetime(2009, 4, 13), 2.8],
            [datetime(2009, 5, 3), 2.1],
            [datetime(2009, 5, 26), 1.1],
            [datetime(2009, 6, 9), 0.25],
            [datetime(2009, 6, 12), 0]
        ]
    }
])

timeseries_plot = PlotView(highcharts_object=timeseries_plot_object,
                        width='500px',
                        height='500px')

### Area Range Plot
averages = [
    [datetime(2009, 7, 1), 21.5], [datetime(2009, 7, 2), 22.1],
    [datetime(2009, 7, 3), 23], [datetime(2009, 7, 4), 23.8],
    [datetime(2009, 7, 5), 21.4], [datetime(2009, 7, 6), 21.3],
```

```

[datetime(2009, 7, 7), 18.3], [datetime(2009, 7, 8), 15.4],
[datetime(2009, 7, 9), 16.4],[datetime(2009, 7, 10), 17.7],
[datetime(2009, 7, 11), 17.5],
[datetime(2009, 7, 12), 17.6],[datetime(2009, 7, 13), 17.7],
[datetime(2009, 7, 14), 16.8], [datetime(2009, 7, 15), 17.7],
[datetime(2009, 7, 16), 16.3], [datetime(2009, 7, 17), 17.8],
[datetime(2009, 7, 18), 18.1],[datetime(2009, 7, 19), 17.2],
[datetime(2009, 7, 20), 14.4]

]

ranges = [
[datetime(2009, 7, 1), 14.3, 27.7], [datetime(2009, 7, 2), 14.5, 27.8],
[datetime(2009, 7, 3),[datetime(2009, 7, 4), 16.7, 30.7],
[datetime(2009, 7, 5), 16.5, 25.0],
[datetime(2009, 7, 6),[datetime(2009, 7, 7), 13.5, 24.8],
[datetime(2009, 7, 8), 10.5, 21.4],
[datetime(2009, 7, 9),[datetime(2009, 7, 10), 11.6, 21.8],
[datetime(2009, 7, 11), 10.7, 23.7],
[datetime(2009, 7,[datetime(2009, 7, 13), 11.6, 23.7],
[datetime(2009, 7, 14), 11.8, 20.7],
[datetime(2009, 7,[datetime(2009, 7, 31), 10.8, 16.1]

]

area_range_plot_object = HighChartsAreaRange(
    title='July Temperatures',
    y_axis_title='Temperature',
    y_axis_units='*C',
    series=[{
        'name': 'Temperature',
        'data': averages,
        'zIndex': 1,
        'marker': {
            'lineWidth': 2,
        }
    }, {
        'name': 'Range',
        'data': ranges,
        'type': 'arearange',
        'lineWidth': 0,
        'linkedTo': ':previous',
        'fillOpacity': 0.3,
        'zIndex': 0
    }
    ])

area_range_plot = PlotView(highcharts_object=area_range_plot_object,
                           width='500px',
                           height='500px')

### Custom Plot derived from: http://www.highcharts.com/demo/columnrange
custom_plot_dictionary = {
    'chart': {
        'type': 'columnrange',
        'inverted': True
    },

```

```
'title': {
  'text': 'Temperature variation by month'
},
'subtitle': {
  'text': 'Observed in Vik i Sogn, Norway, 2009'
},
'xAxis': {
  'categories': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug',
  'Sep', 'Oct', 'Nov', ],
'yAxis': {
  'title': {
    'text': 'Temperature ( °C )'
  }
},
'tooltip': {
  'valueSuffix': '°C'
},
'plotOptions': {
  'columnrange': {
    'dataLabels': {
      'enabled': True,
      'formatter': "function () {return this.y + '°C'}"
    }
  }
},
'legend': {
  'enabled': False
},
'series': [{
  'name': 'Temperatures',
  'data': [
    [-9.7, 9.4],
    [-8.7, 6.5],
    [-3.5, 9.4],
    [-1.4, 19.9],
    [0.0, 22.6],
    [2.9, 29.5],
    [9.2, 30.7],
    [7.3, 26.5],
    [4.4, 18.0],
    [-3.1, 11.4],
    [-5.2, 10.4],
    [-13.5, 9.8]
  ]
}]
}
```

```
custom_plot = PlotView(highcharts_object=custom_plot_dictionary,
                        width='500px',
                        height='500px')
```

```
# TEMPLATE
```

```
{% gizmo highcharts_plot_view line_plot_view %}
{% gizmo highcharts_plot_view scatter_plot_view %}
{% gizmo highcharts_plot_view web_plot %}
{% gizmo highcharts_plot_view pie_plot_view %}
```

```
{% gizmo highcharts_plot_view bar_plot_view %}
{% gizmo highcharts_plot_view timeseries_plot %}
{% gizmo highcharts_plot_view area_range_plot %}
{% gizmo highcharts_plot_view custom_plot %}
```

High Charts Objects

Use one of these predefined objects to simplify creation of Highcharts plot objects for the `highcharts_plot` parameter of Plot View.

```
class tethys_gizmos.gizmo_options.HighChartsLinePlot (series, title='', subtitle='',
                                                    spline=False, x_axis_title='',
                                                    x_axis_units='', y_axis_title='',
                                                    y_axis_units='', **kwargs)
```

Used to create Highcharts line plot visualizations.

series

list, required

A list of highcharts series dictionaries.

title

str

Title of the plot.

subtitle

str

Subtitle of the plot.

spline

bool

If True, lines are smoothed using a spline technique.

x_axis_title

str

Title of the x-axis.

x_axis_units

str

Units of the x-axis.

y_axis_title

str

Title of the y-axis.

y_axis_units

str

Units of the y-axis.

Example

```
# CONTROLLER
from tethys_apps.sdk.gizmos import HighChartsLinePlot, PlotView

highcharts_object = HighChartsLinePlot(title='Plot Title',
```

```

        subtitle='Plot Subtitle',
        spline=True,
        x_axis_title='Altitude',
        x_axis_units='km',
        y_axis_title='Temperature',
        y_axis_units='°C',
        series=[
            {
                'name': 'Air Temp',
                'color': '#0066ff',
                'marker': {'enabled': False},
                'data': [
                    [0, 5], [10, -70],
                    [20, -86.5], [30, -66.5],
                    [40, -32.1],
                    [50, -12.5], [60, -47.7],
                    [70, -85.7], [80, -106.5]
                ]
            },
            {
                'name': 'Water Temp',
                'color': '#ff6600',
                'data': [
                    [0, 15], [10, -50],
                    [20, -56.5], [30, -46.5],
                    [40, -22.1],
                    [50, -2.5], [60, -27.7],
                    [70, -55.7], [80, -76.5]
                ]
            }
        ]
    )

    line_plot_view = PlotView(highcharts_object=highcharts_object,
                              width='500px',
                              height='500px')

    # TEMPLATE

    {% gizmo plot_view line_plot_view %}

```

```

class tethys_gizmos.gizmo_options.HighChartsScatterPlot (series=[], title='', subtitle='',
                                                         x_axis_title='', x_axis_units='',
                                                         y_axis_title='', y_axis_units='', **kwargs)

```

Use to create a Highcharts scatter plot visualization.

series

list, required

A list of highcharts series dictionaries.

title

str

Title of the plot.

subtitle

str

Subtitle of the plot.

spline

bool

If True, lines are smoothed using a spline technique.

x_axis_title

str

Title of the x-axis.

x_axis_units

str

Units of the x-axis.

y_axis_title

str

Title of the y-axis.

y_axis_units

str

Units of the y-axis.

Example

```
# CONTROLLER
from tethys_apps.sdk.gizmos import HighChartsScatterPlot, PlotView

male_dataset = {
    'name': 'Male',
    'color': '#0066ff',
    'data': [
        [174.0, 65.6], [175.3, 71.8], [193.5, 80.7], [186.5, 72.6],
        [187.2, 78.8], [181.5, 74.8], [184.0, 86.4], [184.5, 78.4],
        [175.0, 62.0], [184.0, 81.6], [180.0, 76.6], [177.8, 83.6],
        [192.0, 90.0], [176.0, 74.6], [174.0, 71.0], [184.0, 79.6],
        [192.7, 93.8], [171.5, 70.0], [173.0, 72.4], [176.0, 85.9],
        [176.0, 78.8], [180.5, 77.8], [172.7, 66.2], [176.0, 86.4],
        [173.5, 81.8], [178.0, 89.6], [180.3, 82.8], [180.3, 76.4],
        [164.5, 63.2], [173.0, 60.9], [183.5, 74.8], [175.5, 70.0],
        [188.0, 72.4], [189.2, 84.1], [172.8, 69.1], [170.0, 59.5],
        [182.0, 67.2], [170.0, 61.3], [177.8, 68.6], [184.2, 80.1],
        [186.7, 87.8], [171.4, 84.7], [172.7, 73.4], [175.3, 72.1],
        [180.3, 82.6], [182.9, 88.7], [188.0, 84.1], [177.2, 94.1],
        [172.1, 74.9], [167.0, 59.1], [169.5, 75.6], [174.0, 86.2],
        [172.7, 75.3], [182.2, 87.1], [164.1, 55.2], [163.0, 57.0],
        [171.5, 61.4], [184.2, 76.8], [174.0, 86.8], [174.0, 72.2],
        [177.0, 71.6], [186.0, 84.8], [167.0, 68.2], [171.8, 66.1]
    ]
}

female_dataset = {
    'name': 'Female',
    'color': '#ff6600',
    'data': [
        [161.2, 51.6], [167.5, 59.0], [159.5, 49.2], [157.0, 63.0],
        [155.8, 53.6], [170.0, 59.0], [159.1, 47.6], [166.0, 69.8],
        [176.2, 66.8], [160.2, 75.2], [172.5, 55.2], [170.9, 54.2],
    ]
}
```

```
[172.9, 62.5], [153.4, 42.0], [160.0, 50.0], [147.2, 49.8],
[168.2, 49.2], [175.0, 73.2], [157.0, 47.8], [167.6, 68.8],
[159.5, 50.6], [175.0, 82.5], [166.8, 57.2], [176.5, 87.8],
[170.2, 72.8], [174.0, 54.5], [173.0, 59.8], [179.9, 67.3],
[170.5, 67.8], [160.0, 47.0], [154.4, 46.2], [162.0, 55.0],
[176.5, 83.0], [160.0, 54.4], [152.0, 45.8], [162.1, 53.6],
[170.0, 73.2], [160.2, 52.1], [161.3, 67.9], [166.4, 56.6],
[168.9, 62.3], [163.8, 58.5], [167.6, 54.5], [160.0, 50.2],
[161.3, 60.3], [167.6, 58.3], [165.1, 56.2], [160.0, 50.2],
[170.0, 72.9], [157.5, 59.8], [167.6, 61.0], [160.7, 69.1],
[163.2, 55.9], [152.4, 46.5], [157.5, 54.3], [168.3, 54.8],
[180.3, 60.7], [165.5, 60.0], [165.0, 62.0], [164.5, 60.3]
]
}

highcharts_object = HighChartsScatterPlot(title='Scatter Plot',
                                         subtitle='Scatter Plot',
                                         x_axis_title='Height',
                                         x_axis_units='cm',
                                         y_axis_title='Weight',
                                         y_axis_units='kg',
                                         series=[male_dataset, female_dataset]
)

scatter_plot_view = PlotView(highcharts_object=highcharts_object,
                             width='500px',
                             height='500px')

# TEMPLATE

{% gizmo plot_view scatter_plot_view %}
```

```
class tethys_gizmos.gizmo_options.HighChartsPolarPlot(series=[], title='', subtitle='',
                                                    categories=[], **kwargs)
```

Use to create a Highcharts polar plot visualization.

series

list, required

A list of highcharts series dictionaries.

title

str

Title of the plot.

subtitle

str

Subtitle of the plot.

categories

list

List of category names, one for each data point in the series.

Example

```
# CONTROLLER
from tethys_apps.sdk.gizmos import HighChartsPolarPlot, PlotView
```

```

web_plot_object = HighChartsPolarPlot(title='Polar Chart',
    subtitle='Polar Chart',
    pane={
        'size': '80%'
    },
    categories=['Infiltration', 'Soil Moisture', 'Precipitation',
        'Evaporation',
        'Roughness', 'Runoff', 'Permeability', 'Vegetation'],
    series=[
        {
            'name': 'Park City',
            'data': [0.2, 0.5, 0.1, 0.8, 0.2, 0.6, 0.8, 0.3],
            'pointPlacement': 'on'
        },
        {
            'name': 'Little Dell',
            'data': [0.8, 0.3, 0.2, 0.5, 0.1, 0.8, 0.2, 0.6],
            'pointPlacement': 'on'
        }
    ]
)

web_plot = PlotView(highcharts_object=web_plot_object,
    width='500px',
    height='500px')

```

```
# TEMPLATE
```

```
{% gizmo plot_view web_plot %}
```

```
class tethys_gizmos.gizmo_options.HighChartsPiePlot(series=[], title='', subtitle='',
    **kwargs)
```

Use to create a Highcharts pie plot visualization.

series

list, required

A list of highcharts series dictionaries.

title

str

Title of the plot.

subtitle

str

Subtitle of the plot.

Example:

```

# CONTROLLER
from tethys_apps.sdk.gizmos import HighChartsPieChart, PlotView

pie_plot_object = HighChartsPiePlot(title='Pie Chart',
    subtitle='Pie Chart',
    series=[{
        'type': 'pie',
        'name': 'Browser share',
        'data': [
            ['Firefox', 45.0],

```



```
        ['IE', 26.8],
        {
            'name': 'Chrome',
            'y': 12.8,
            'sliced': True,
            'selected': True
        },
        ['Safari', 8.5],
        ['Opera', 6.2],
        ['Others', 0.7]
    ]
}

)

pie_plot_view = PlotView(highcharts_object=pie_plot_object,
                        width='500px',
                        height='500px')

# TEMPLATE

{% gizmo plot_view pie_plot_view %}
```

```
class tethys_gizmos.gizmo_options.HighChartsBarPlot (series=[], title='', subtitle='', hor-
            izontal=False, categories=[],
            axis_title='', axis_units='',
            group_tools=True, **kwargs)
```

Bar Plot

Displays as either a bar or column chart.

series

list, required

A list of highcharts series dictionaries.

title

str

Title of the plot.

subtitle

str

Subtitle of the plot.

horizontal

bool

If True, bars are displayed horizontally, otherwise they are displayed vertically.

categories

list

A list of category titles, one for each bar.

axis_title

str

Title of the axis.

axis_units

str

Units of the axis.

Example

```
# CONTROLLER
from tethys_apps.sdk.gizmos import HighChartsBarPlot, PlotView

bar_plot_view = HighChartsBarPlot(
    title='Bar Chart',
    subtitle='Bar Chart',
    vertical=False,
    categories=[
        'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct',
        'Nov', 'Dec' ],
    axis_units='millions',
    axis_title='Population',
    series=[{
        'name': 'Year 1800',
        'data': [107, 31, 635, 203, 2]
    }, {
        'name': 'Year 1900',
        'data': [133, 156, 947, 408, 6]
    }, {
        'name': 'Year 2008',
        'data': [973, 914, 4054, 732, 34]}
    ]
)

bar_plot_view = PlotView(highcharts_object=bar_plot_view,
    width='500px',
    height='500px')

# TEMPLATE

{% gizmo plot_view bar_plot_view %}
```

```
class tethys_gizmos.gizmo_options.HighChartsTimeSeries (series=[], title='', subtitle='',
    y_axis_title='', y_axis_units='', **kwargs)
```

Use to create a Highcharts timeseries plot visualization

series

list, required

A list of highcharts series dictionaries.

title

str

Title of the plot.

subtitle

str

Subtitle of the plot.

y_axis_title

str

Title of the axis.

y_axis_units*str*

Units of the axis.

Example

```
# CONTROLLER
from tethys_apps.sdk.gizmos import HighChartsTimeSeries, PlotView

timeseries_plot_object = HighChartsTimeSeries(
    title='Irregular Timeseries Plot',
    y_axis_title='Snow depth',
    y_axis_units='m',
    series=[{
        'name': 'Winter 2007-2008',
        'data': [
            [datetime(2008, 12, 2), 0.8],
            [datetime(2008, 12, 9), 0.6],
            [datetime(2008, 12, 16), 0.6],
            [datetime(2008, 12, 28), 0.67],
            [datetime(2009, 1, 1), 0.81],
            [datetime(2009, 1, 8), 0.78],
            [datetime(2009, 1, 12), 0.98],
            [datetime(2009, 1, 27), 1.84],
            [datetime(2009, 2, 10), 1.80],
            [datetime(2009, 2, 18), 1.80],
            [datetime(2009, 2, 24), 1.92],
            [datetime(2009, 3, 4), 2.49],
            [datetime(2009, 3, 11), 2.79],
            [datetime(2009, 3, 15), 2.73],
            [datetime(2009, 3, 25), 2.61],
            [datetime(2009, 4, 2), 2.76],
            [datetime(2009, 4, 6), 2.82],
            [datetime(2009, 4, 13), 2.8],
            [datetime(2009, 5, 3), 2.1],
            [datetime(2009, 5, 26), 1.1],
            [datetime(2009, 6, 9), 0.25],
            [datetime(2009, 6, 12), 0]
        ]
    }]
)

timeseries_plot = PlotView(highcharts_object=timeseries_plot_object,
    width='500px',
    height='500px')

# TEMPLATE

{% gizmo plot_view timeseries_plot %}

class tethys_gizmos.gizmo_options.HighChartsAreaRange(series=[], title='', subtitle='',
    y_axis_title='', y_axis_units='',
    **kwargs)
```

Use to create a Highcharts area range plot visualization.

series*list, required*

A list of highcharts series dictionaries.

title*str*

Title of the plot.

subtitle*str*

Subtitle of the plot.

y_axis_title*str*

Title of the axis.

y_axis_units*str*

Units of the axis.

Example

```
# CONTROLLER
from tethys_apps.sdk.gizmos import HighChartsAreaRange, PlotView

averages = [
    [datetime(2009, 7, 1), 21.5], [datetime(2009, 7, 2), 22.1],
    [datetime(2009, 7, 4), 23.8], [datetime(2009, 7, 5), 21.4],
    [datetime(2009, 7, 7), 18.3], [datetime(2009, 7, 8), 15.4],
    [datetime(2009, 7, 10), 17.7], [datetime(2009, 7, 11), 17.5],
    [datetime(2009, 7, 12), 17.6], [datetime(2009, 7, 13), 17.7],
    [datetime(2009, 7, 14), 16.8], [datetime(2009, 7, 15), 17.7],
    [datetime(2009, 7, 16), 16.3], [datetime(2009, 7, 17), 17.8],
    [datetime(2009, 7, 18), 18.1], [datetime(2009, 7, 19), 17.2],
    [datetime(2009, 7, 20), 14.4]
]

ranges = [
    [datetime(2009, 7, 1), 14.3, 27.7], [datetime(2009, 7, 2), 14.5, 27.8],
    [datetime(2009, 7, 3), [datetime(2009, 7, 4), 16.7, 30.7],
    [datetime(2009, 7, 5), 16.5, 25.0], [datetime(2009, 7, 6),
    [datetime(2009, 7, 7), 13.5, 24.8], [datetime(2009, 7, 8), 10.5, 21.4],
    [datetime(2009, 7, 9),
    [datetime(2009, 7, 10), 11.6, 21.8], [datetime(2009, 7, 11), 10.7, 23.7],
    [datetime(2009, 7, [datetime(2009, 7, 13), 11.6, 23.7],
    [datetime(2009, 7, 14), 11.8, 20.7],
    [datetime(2009, 7, [datetime(2009, 7, 16), 13.6, 19.6],
    [datetime(2009, 7, 17), 11.4, 22.6],
    [datetime(2009, 7, [datetime(2009, 7, 19), 14.2, 21.6]
]

area_range_plot_object = HighChartsAreaRange(
    title='July Temperatures',
    y_axis_title='Temperature',
    y_axis_units='*C',
    series=[{
        'name': 'Temperature',
        'data': averages,
        'zIndex': 1,
    }
    ]
)
```

```
        'marker': {
            'lineWidth': 2,
        }
    }, {
        'name': 'Range',
        'data': ranges,
        'type': 'arearange',
        'lineWidth': 0,
        'linkedTo': ':previous',
        'fillOpacity': 0.3,
        'zIndex': 0
    }
]
)

area_range_plot = PlotView(highcharts_object=area_range_plot_object,
                           width='500px',
                           height='500px')

# TEMPLATE

{% gizmo plot_view area_range_plot %}
```

JavaScript API

The Highcharts plots can be modified via JavaScript by using jQuery to select the Highcharts div and calling the `highcharts()` method on it. This will return the JavaScript object that represents the plot, which can be modified using the [Highcharts API](#).

```
var plot = $('#my-plot').highcharts();
```

Map View

```
class tethys_gizmos.gizmo_options.MapView (height='100%', width='100%',  
                                             basemap='OpenStreetMap', view={'center':  
                                             [-100, 40], 'zoom': 2}, controls=[], lay-  
                                             ers=[], draw=None, legend=False, attributes=''  
                                             classes='')
```

The Map View gizmo can be used to produce interactive maps of spatial data. It is powered by OpenLayers 3, a free and open source pure javascript mapping library. It supports layers in a variety of different formats including WMS, Tiled WMS, GeoJSON, KML, and ArcGIS REST. It includes drawing capabilities and the ability to create a legend for the layers included in the map.

Shapes that are drawn on the map by users can be retrieved from the map via a hidden text field named 'geometry' and it is updated every time the map is changed. The text in the text field is a string representation of JSON. The geometry definition contained in this JSON can be formatted as either GeoJSON or Well Known Text. This can be configured via the `output_format` option of the `MVDraw` object. If the Map View is embedded in a form, the geometry that is drawn on the map will automatically be submitted with the rest of the form via the hidden text field.

height

str

Height of the map element. Any valid css unit of length (e.g.: '500px'). Defaults to '520px'.

width

str

Width of the map element. Any valid css unit of length (e.g.: '100%'). Defaults to '100%'.

basemap

str or dict

The base map to display: either OpenStreetMap, MapQuest, or a Bing map. Valid values for the string option are: 'OpenStreetMap' and 'MapQuest'. If you wish to configure the base map with options, you must use the dictionary form. The dictionary form is required to use a Bing map, because an API key must be passed as an option. See below for more detail.

view

MVView

An MVView object specifying the initial view or extent for the map.

controls

list

A list of controls to add to the map. The list can be a list of strings or a list of dictionaries. Valid controls are ZoomSlider, Rotate, FullScreen, ScaleLine, ZoomToExtent, and 'MousePosition'. See below for more detail.

layers

list

A list of MVLayer objects.

draw

MVDraw

An MVDraw object specifying the drawing options.

attributes

str

A string representing additional HTML attributes to add to the primary element (e.g. "onclick=run_me();").

classes

str

Additional classes to add to the primary HTML element (e.g. "example-class another-class").

Options Dictionaries

Many of the options above will accept dictionaries with additional options. These dictionaries should be structured with a single key that is the name of the original option with a value of another dictionary containing the additional options. For example, to provide additional options for the 'ZoomToExtent' control, you would create a dictionary with key 'ZoomToExtent' and value of a dictionary with the additional options like this:

```
{'ZoomToExtent': {'projection': 'EPSG:4326', 'extent': [-135, 22, -55, 54]}}
```

Most of the additional options correspond with the options objects in the OpenLayers API. The following sections provide links to the OpenLayers objects that you can refer to when selecting the options.

Base Maps

There are three base maps supported by the Map View gizmo: OpenStreetMap, Bing, and MapQuest. Use the following links to learn about the additional options you can configure the base maps with:

- Bing: [ol.source.BingMaps](#)
- MapQuest: [ol.source.MapQuest](#)
- OpenStreetMap: [ol.source.OSM](#)

```
{'Bing': {'key': 'Ap|k3yheRE', 'imagerySet': 'Aerial'}}
```

Controls

Use the following links to learn about options for the different controls:

- Fullscreen: [ol.control.FullScreen](#)
- MousePosition: [ol.control.MousePosition](#)
- Rotate: [ol.control.Rotate](#)
- ScaleLine: [ol.control.ScaleLine](#)
- ZoomSlider: [ol.control.ZoomSlider](#)
- ZoomToExtent: [ol.control.ZoomToExtent](#)

Example

```
# CONTROLLER
from tethys_apps.sdk.gizmos import MapView, MVDraw, MVView, MVLayer, MVLegendClass

# Define view options
view_options = MVView(
    projection='EPSG:4326',
    center=[-100, 40],
    zoom=3.5,
    maxZoom=18,
    minZoom=2
)

# Define drawing options
drawing_options = MVDraw(
    controls=['Modify', 'Move', 'Point', 'LineString', 'Polygon', 'Box'],
    initial='Point',
    output_format='WKT'
)

# Define GeoJSON layer
geojson_object = {
    'type': 'FeatureCollection',
    'crs': {
        'type': 'name',
        'properties': {
            'name': 'EPSG:3857'
        }
    },
    'features': [
        {
            'type': 'Feature',
            'geometry': {
                'type': 'Point',
                'coordinates': [0, 0]
            }
        },
        {
            'type': 'Feature',
            'geometry': {
                'type': 'LineString',
                'coordinates': [[4e6, -2e6], [8e6, 2e6]]
            }
        }
    ]
}
```

```

    }
  },
  {
    'type': 'Feature',
    'geometry': {
      'type': 'Polygon',
      'coordinates': [[[-5e6, -1e6], [-4e6, 1e6], [-3e6, -1e6]]]
    }
  }
]
}

geojson_layer = MVLayer(source='GeoJSON',
  options=geojson_object,
  legend_title='Test GeoJSON',
  legend_extent=[-46.7, -48.5, 74, 59],
  legend_classes=[
    MVLegendClass('polygon', 'Polygons', fill='rgba(255,255,255,0.8)',
      stroke='#3d9dcd'), MVLegendClass('line', 'Lines', stroke='#3d9dcd')
  ])

# Define GeoServer Layer
geoserver_layer = MVLayer(source='ImageWMS',
  options={'url': 'http://192.168.59.103:8181/geoserver/wms',
    'params': {'LAYERS': 'topp:states'},
    'serverType': 'geoserver'},
  legend_title='USA Population',
  legend_extent=[-126, 24.5, -66.2, 49],
  legend_classes=[
    MVLegendClass('polygon', 'Low Density', fill='#00ff00', stroke='#000000'),
    MVLegendClass('polygon', 'Medium Density', fill='#ff0000', stroke='#000000'),
    MVLegendClass('polygon', 'High Density', fill='#0000ff', stroke='#000000')
  ])

# Define KML Layer
kml_layer = MVLayer(source='KML',
  options={'url': '/static/tethys_gizmos/data/model.kml'},
  legend_title='Park City Watershed',
  legend_extent=[-111.60, 40.57, -111.43, 40.70],
  legend_classes=[
    MVLegendClass('polygon', 'Watershed Boundary', fill='#ff8000'),
    MVLegendClass('line', 'Stream Network', stroke='#0000ff'),
  ])

# Tiled ArcGIS REST Layer
arc_gis_layer = MVLayer(source='TileArcGISRest',
  options={'url': 'http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/'
    legend_title='ESRI USA Highway',
    legend_extent=[-173, 17, -65, 72])

# Define map view options
map_view_options = MapView(
  height='600px',
  width='100%',
  controls=['ZoomSlider', 'Rotate', 'FullScreen',
    {'MousePosition': {'projection': 'EPSG:4326'}},
    {'ZoomToExtent': {'projection': 'EPSG:4326',
      'extent': [-130, 22, -65, 54]}}],
  layers=[geojson_layer, geoserver_layer, kml_layer, arc_gis_layer],

```



```
        view=view_options,
        basemap='OpenStreetMap',
        draw=drawing_options,
        legend=True
    )

# TEMPLATE

{% gizmo map_view map_view_options %}
```

Secondary

```
class tethys_gizmos.gizmo_options.MVLayer(source, options, legend_title, legend_classes=None, legend_extent=None, legend_extent_projection='EPSG:4326')
```

MVLayer objects are used to define map layers for the Map View Gizmo.

source

str, required

The source or data type of the layer (e.g.: ImageWMS)

options

dict, required

A dictionary representation of the OpenLayers layer options object for the source.

legend_title

str, required

The human readable name of the layer that will be displayed in the legend.

legend_classes

list

A list of MVLegendClass objects.

legend_extent

list

A list of four ordinates representing the extent that will be used on “zoom to layer”: [minx, miny, maxx, maxy].

legend_extent_projection

str

The EPSG projection of the extent coordinates. Defaults to “EPSG:4326”.

Example

```
# Define GeoJSON layer
geojson_object = {
    'type': 'FeatureCollection',
    'crs': {
        'type': 'name',
        'properties': {
            'name': 'EPSG:3857'
        }
    },
    'features': [
        {
```

```

        'type': 'Feature',
        'geometry': {
            'type': 'Point',
            'coordinates': [0, 0]
        }
    },
    {
        'type': 'Feature',
        'geometry': {
            'type': 'LineString',
            'coordinates': [[4e6, -2e6], [8e6, 2e6]]
        }
    },
    {
        'type': 'Feature',
        'geometry': {
            'type': 'Polygon',
            'coordinates': [[[-5e6, -1e6], [-4e6, 1e6], [-3e6, -1e6]]]
        }
    }
]
}

geojson_layer = MVLayer(source='GeoJSON',
    options=geojson_object,
    legend_title='Test GeoJSON',
    legend_extent=[-46.7, -48.5, 74, 59],
    legend_classes=[
        MVLegendClass('polygon', 'Polygons', fill='rgba(255,255,255,0.8)', stroke
        MVLegendClass('line', 'Lines', stroke='#3d9dcd')
    ])

# Define GeoServer Layer
geoserver_layer = MVLayer(source='ImageWMS',
    options={'url': 'http://192.168.59.103:8181/geoserver/wms',
            'params': {'LAYERS': 'topp:states'},
            'serverType': 'geoserver'},
    legend_title='USA Population',
    legend_extent=[-126, 24.5, -66.2, 49],
    legend_classes=[
        MVLegendClass('polygon', 'Low Density', fill='#00ff00', stroke='#000000')
        MVLegendClass('polygon', 'Medium Density', fill='#ff0000', stroke=')
        MVLegendClass('polygon', 'High Density', fill='#0000ff', stroke='#000000')
    ])

# Define KML Layer
kml_layer = MVLayer(source='KML',
    options={'url': '/static/tethys_gizmos/data/model.kml'},
    legend_title='Park City Watershed',
    legend_extent=[-111.60, 40.57, -111.43, 40.70],
    legend_classes=[
        MVLegendClass('polygon', 'Watershed Boundary', fill='#ff8000'),
        MVLegendClass('line', 'Stream Network', stroke='#0000ff'),
    ])

# Tiled ArcGIS REST Layer
arc_gis_layer = MVLayer(source='TileArcGISRest',
    options={'url': 'http://sampleserver1.arcgisonline.com/ArcGIS/rest/services/

```

```
legend_title='ESRI USA Highway',  
legend_extent=[-173, 17, -65, 72]),
```

class `tethys_gizmos.gizmo_options.MVLegendClass` (*type*, *value*, *fill*='', *stroke*='', *ramp*=[])

MVLegendClasses are used to define the classes listed in the legend.

type

str, *required*

The type of feature to be represented by the legend class. Either 'point', 'line', 'polygon', or 'raster'.

value

str, *required*

The value or name of the legend class.

fill

str

Valid RGB color for the fill (e.g.: '#00ff00', 'rgba(0, 255, 0, 0.5)'). Required for 'point' or 'polygon' types.

stroke

str

Valid RGB color for the stroke/line (e.g.: '#00ff00', 'rgba(0, 255, 0, 0.5)'). Required for 'line' types and optional for 'polygon' types.

ramp

list

A list of hexadecimal RGB colors that will be used to construct a color ramp. Required for 'raster' types.

Example

```
point_class = MVLegendClass(type='point', value='Cities', fill='#00ff00')  
line_class = MVLegendClass(type='line', value='Roads', stroke='rgba(0,0,0,0.7)')  
polygon_class = MVLegendClass(type='polygon', value='Lakes', stroke='#0000aa',  
fill='#0000ff')
```

class `tethys_gizmos.gizmo_options.MVDraw` (*controls*, *initial*, *output_format*='GeoJSON')

MVDraw objects are used to define the drawing options for Map View.

controls

list, *required*

List of drawing controls to add to the map. Valid options are 'Modify', 'Move', 'Point', 'LineString', 'Polygon' and 'Box'.

initial

str, *required*

Drawing control to be enabled initially. Must be included in the controls list.

output_format

str

Format to output to the hidden text area. Either 'WKT' (for Well Known Text format) or 'GeoJSON'. Defaults to 'GeoJSON'

Example

```
drawing_options = MVDraw(  
    controls=['Modify', 'Move', 'Point', 'LineString', 'Polygon', 'Box'],  
    initial='Point',
```

```

    output_format='WKT'
)

```

class `tethys_gizmos.gizmo_options.MVView`(*projection*, *center*, *zoom*, *maxZoom=28*, *minZoom=0*)

MVView objects are used to define the initial view of the Map View. The initial view is set by specifying a center and a zoom level.

projection

str

Projection of the center coordinates given. This projection will be used to transform the coordinates into the default map projection (EPSG:3857).

center

list

An array with the coordinates of the center point of the initial view.

zoom

int or float

The zoom level for the initial view.

maxZoom

int or float

The maximum zoom level allowed. Defaults to 28.

minZoom

int or float

The minimum zoom level allowed. Defaults to 0.

Example

```

view_options = MVView(
    projection='EPSG:4326',
    center=[-100, 40],
    zoom=3.5,
    maxZoom=18,
    minZoom=2
)

```

JavaScript API

For advanced features, the JavaScript API can be used to interact with the OpenLayers map object that is generated by the Map View JavaScript library.

TETHYS_MAP_VIEW.getMap() This method returns the OpenLayers map object. You can use the [OpenLayers Map API](#) to perform operations on this object such as adding layers and custom controls.

```

var ol_map = TETHYS_MAP_VIEW.map;
ol_map.addLayer(...);
ol_map.setView(...);

```

Google Map View

```
class tethys_gizmos.gizmo_options.GoogleMapView(height, width, maps_api_key='',
                                                reference_kml_action='', drawing_types_enabled=[],
                                                initial_drawing_mode='', output_format='GEOJSON',
                                                input_overlays=[None], attributes='',
                                                classes='')
```

Google Map View

The Google Map View is similar to Map View, but it is powered by Google Maps 3. It has the drawing library enabled to allow geospatial user input. An optional background dataset can be specified for reference, but only the shapes drawn by the user are returned (see [Retrieving Shapes reference](#) section).

Shapes that are drawn on the map by users can be retrieved from the map in two ways. A hidden text field named 'geometry' is updated every time the map is changed. The text in the text field is a string representation of JSON. The geometry can be formatted as either GeoJSON or Well Known Text. This can be configured by setting the output_format parameter. If the Google Map View is embedded in a form, the geometry that is drawn on the map will automatically be submitted with the rest of the form via the hidden text field.

Alternatively, the data can be extracted directly using the JavaScript API (see below).

height

string, required

Height of map container in normal css units

width

string, required

Width of map container in normal css units

maps_api_key

string, required

The Google Maps API key. If the API key is provided in the settings.py via the TETHYS_GIZMOS_GOOGLE_MAPS_API_KEY option, this parameter is not required.

reference_kml_action

url string

The action that returns the background kml datasets. These datasets are used for reference only.

drawing_types_enabled

list of strings

A list of the types of geometries the user will be allowed to draw (POLYGONS, POINTS, POLYLINES).

initial_drawing_mode

string

A string representing the drawing mode that will be enabled by default. Valid modes are: 'POLYGONS', 'POINTS', 'POLYLINES'. The mode used must be one of the drawing_types_enabled that the user is allowed to draw.

output_format

string

A string specifying the format of the string that is output by the editable map tool. Valid values are 'GEOJSON' for GeoJSON format or 'WKT' for Well Known Text Format.

input_overlays*PySON*

A JavaScript-equivalent Python data structure representing GeoJSON or WktJSON containing the geometry and attributes to be added to the map as overlays (see example below). Only points, lines and polygons are supported.

attributes*str*

A string representing additional HTML attributes to add to the primary element (e.g. “onclick=run_me();”).

classes*str*

Additional classes to add to the primary HTML element (e.g. “example-class another-class”).

Example

```
# CONTROLLER
from tethys_apps.sdk.gizmos import GoogleMapView

google_map_view = GoogleMapView(height='600px',
                                width='100%',
                                reference_kml_action=reverse('gizmos:get_kml'),
                                drawing_types_enabled=['POLYGONS', 'POINTS', 'POLYLINES'],
                                initial_drawing_mode='POINTS',
                                output_format='WKT')

# GeoJSON Example
geo_json = {'type': 'WKTGeometryCollection',
            'geometries': [
                {'type': 'Point',
                 'wkt': 'POINT(-111.5123462677002 40.629197012613545)',
                 'properties': {'id': 1, 'value': 1}
                },
                {'type': 'Polygon',
                 'wkt': 'POLYGON((-111.50153160095215 40.63193284946615,
                                   -111.50101661682129 40.617210120505035, 'properties': {'id': 2, 'value': 2}
                                   )
                },
                {'type': 'PolyLine', #
                 'wkt': 'POLYLINE(-111.49123191833496 40.65003865742191,
                                   -111.49088859558105 40.635319920747456, 'properties': {'id': 3, 'value': 3}
                                   )
                }
            ]
            }

google_map_view_options = {'height': '700px',
                           'width': '100%',
                           'maps_api_key': 'S0mEaPIk3y',
                           'drawing_types_enabled': ['POLYGONS', 'POINTS', 'POLYLINES'],
                           'initial_drawing_mode': 'POINTS',
                           'input_overlays': geo_json}

# WKT Example
wkt_json = {"type": "GeometryCollection",
            "geometries": [
                {"type": "Point",
```

```
    "coordinates": [40.629197012613545, -111.5123462677002],
    "properties": {"id": 1, "value": 1},
    {"type": "Polygon",
     "coordinates": [[40.63193284946615, -111.50153160095215],
     [40.617210120505035, -111.50101661682129],
     [40.623594711231775, -111.48625373840332],
     [40.63193284946615, -111.49123191833496]], "properties": {"id": 2, "value": 2}},
    {"type": "LineString",
     "coordinates": [[40.65003865742191, -111.49123191833496],
     [40.635319920747456, -111.49088859558105],
     [40.64912697157757, -111.48127555847168],
     [40.634668574229735, -111.48024559020996]], "properties": {"id": 3, "value": 3}}
  ]}

google_map_view_options = {'height': '700px',
                           'width': '100%',
                           'maps_api_key': 'S0mEaPIk3y',
                           'drawing_types_enabled': ['POLYGONS', 'POINTS', 'POLYLINES'],
                           'initial_drawing_mode': 'POINTS',
                           'input_overlays': wkt_json}

# TEMPLATE

{% gizmo google_map_view google_map_view_options %}
```

JavaScript API

For advanced features, the JavaScript API can be used to interact with the editable map. If you need capabilities beyond the scope of this API, we recommend using the Google Maps version 3 API to create your own map.

TETHYS_GOOGLE_MAP_VIEW.getMap() This method returns the Google Map object for direct manipulation through JavaScript.

TETHYS_GOOGLE_MAP_VIEW.getGeoJson() This method returns the GeoJSON object representing all of the overlays on the map.

TETHYS_GOOGLE_MAP_VIEW.getGeoJsonString() This method returns a stringified GeoJSON object representing all of the overlays on the map.

TETHYS_GOOGLE_MAP_VIEW.getWktJson() This method returns a Well Known Text JSON object representing all of the overlays on the map.

TETHYS_GOOGLE_MAP_VIEW.getWktJsonString() This method returns a stringified Well Known Text JSON object representing all of the overlays on the map.

TETHYS_GOOGLE_MAP_VIEW.swapKmlService(kml_service) Use this method to swap out the current reference kml layers for new ones.

- **kml_service** (*string*) = URL endpoint that returns a JSON object with a property called 'kml_link' that is an array of publicly accessible URLs to kml or kmz documents

TETHYS_GOOGLE_MAP_VIEW.swapOverlayService(overlay_service, clear_overlays) Use this method to add new overlays to the map dynamically without reloading the page.

- **overlay_service** (*string*) = URL endpoint that returns a JSON object with a property called 'overlay_json' that has a value of a WKT or GeoJSON object in the same format as is used for input_overlays
- **clear_overlays** (*boolean*) = if true, will clear all overlays from the map prior to adding the new overlays. Otherwise all overlays will be retained.

Fetch Climate Map

```
class tethys_gizmos.gizmo_options.FetchClimateMap(url_parameter={},
                                                map_parameters={},
                                                plot_parameters={},           vari-
                                                able_parameters={},
                                                grid_parameters={},
                                                point_parameters={})
```

Plot Parameters

dimensions

dict

The integer is in pixels for width ([Highcharts width reference](#)) or height ([Highcharts height reference](#)). Not required to be defined.

Example

```
# CONTROLLER
fetchclimate_map = FetchClimateMap(
    url_parameter=FetchClimateURLParameter(
        serverUrl='http://fetchclimate2.cloudapp.net'), variable_parameters =
    FetchClimateVariableParameters(variables={
        'prate': [423, 432, 426, 424],
        'elev': []
    }),
    map_parameters=FetchClimateMapParameters(
        css={'height': '600px',
            'width': '100%'},
        map_data=FetchClimateMapData(
            drawing_types_enabled=['RECTANGLE', 'POINTS'],
            initial_drawing_mode='RECTANGLE',
            max_num_grids=2
        )
    ),
    grid_parameters=FetchClimateGridParameters(
        title='Provo Canyon Watershed',
        boundingBox=[40.308836, 40.381579, -111.654462, -111.550778],
        gridResolution=[25, 25]
    ),
    point_parameters=FetchClimatePointParameters(
        title='Clyde Building',
        location=[40.246942, -111.647971],
    ),
    plot_parameters=FetchClimatePlotParameters(
        dimensions={'width': 500, 'height': 350})
)

# TEMPLATE

{% gizmo fetchclimate fetchclimate_with_map_plot %}
```


Secondary

class `tethys_gizmos.gizmo_options.FetchClimateURLParameter` (
`serverUrl='http://fetchclimate2.cloudapp.net'`)

URL Parameter

This gizmo can be used to get climate data based off of a bounding box over an area or a point. It is based off of Microsoft Research. See the [FetchClimate page reference](#) for more info.

serverUrl

str

The URL to the FetchClimate server (e.g `'serverUrl':'http://fetchclimate2.cloudapp.net'`)

class `tethys_gizmos.gizmo_options.FetchClimateMapData` (`api_key=''`, `drawing_types_enabled=['RECTANGLE']`,
`initial_drawing_mode='RECTANGLE'`,
`max_num_grids=0`,
`max_num_points=0`)

Map Parameters - map_data

api_key

str

API key for Google maps.

drawing_types_enabled

str

A list of the types of geometries the user will be allowed to draw.

Valid types are

RECTANGLE, and POINTS. (e.g.: `drawing_types_enabled=['RECTANGLE','POINTS']`)

initial_drawing_mode

str

A string representing the drawing mode that will be enabled by default.

Valid modes are

'RECTANGLE', 'POINTS'. The mode used must be one of the `drawing_types_enabled` that the user is allowed to draw.

max_num_grids

int

The maximum number of grids allowed for the user. Default is unlimited. (e.g. `'max_num_grids':0`).

max_num_points

int

The maximum number of points allowed for the user. Default is unlimited. (e.g `'max_num_points':0`).

class `tethys_gizmos.gizmo_options.FetchClimateMapParameters` (`css={}`, `map_data={}`)

Map Parameters Optional if grid or point included. Otherwise, required!

css

dict

Custom css elements. FORMAT: `{'css-element-name': 'css-value'}`.

If no width or height included, 500px X 500px assumed.

map_data*dict*

Data needed to create the map.

```
class tethys_gizmos.gizmo_options.FetchClimatePlotParameters (dimensions={'width':
                                                                    100, 'height': 500})
```

Plot Parameters

dimensions*dict*

The integer is in pixels for width ([Highcharts width reference](#)) or height ([Highcharts height reference](#)). Not required to be defined.

```
class tethys_gizmos.gizmo_options.FetchClimateVariableParameters (variables={'precip':
                                                                              []})
```

Variable Parameters

To find out which variables you can use and their parameters, go to your service url with '/api/configuration' at the end. (e.g. <http://fetchclimate2.cloudapp.net/api/configuration>). Look in "EnvironmentalVariables" for the variable names. Then, to find the data source ID's of sources available, go to "DataSources".

variables*dict*

Must have variable defined. It is in the format { 'variable_name':[variable_id,variable_id,variable_id]}.

```
class tethys_gizmos.gizmo_options.FetchClimateGridParameters (title='', bounding-
                                                             Box=[], gridResolu-
                                                             tion=[])
```

Grid Parameters

Optional if there is a map or point included. Otherwise, it is required! No map needed. If map included, it will initialize with input grid.

title*str*

The name of the grid area.

boundingBox*dict*

An array of length 4 with bounding lat and long. e.g.[min lat, max lat, min lon, max long].

gridResolution*dict*

An array of length 2. Number of grid cells in lat and lon directions. e.g.[lat resolution,lon resolution].

```
class tethys_gizmos.gizmo_options.FetchClimatePointParameters (title='', location=[])
```

Point Parameters

Optional if there is a map or grid included. Otherwise, it is required! No map needed. If map included, it will initialize with input point.

title*str*

The name of the point location.

location*dict*

An array of length 2 with lat and lon of point. e.g.[lat,lon].

JavaScript API

For advanced features, the JavaScript API can be used to get the data once the request is complete.

This is also available if you use the plot feature. To use it, replace all 'date' or 'DATE' in the names with 'plot' or 'PLOT' (except for 'fcDataRequestComplete' or 'fcOneDataRequestComplete').

FETCHCLIMATE_DATA.getAllData() ONCE THE AJAX CALLS ARE COMPLETE - This method returns an object with the initial key level as the variable names and the next level as the grid/point names with the data inside of the grid name key. However, if the AJAX calls are not complete, it returns -1.

How to know when the AJAX calls are complete? - The Event Listener

When *ALL* requests are complete - 'e.detail' contains all of the data returned from the AJAX calls:

```
//The Entire Request Complete Event Listener</p>
jQuery('#fetchclimate_data')[0].addEventListener('fcDataRequestComplete', function(e) {
    console.log(e.detail);
});
```

When *ONE* of the requests is complete - 'e.detail' contains all of the data returned from the AJAX calls:

```
//The Single Request Complete Event Listener</p>
jQuery('#fetchclimate_data')[0].addEventListener('fcOneDataRequestComplete', function(e) {
    console.log(e.detail);
});
```

1.6.5 Persistent Stores API

Last Updated: November 12, 2014

The Persistent Store API streamlines the use of SQL databases in Tethys apps. Using this API, you can provision up to 5 SQL databases for your app. The databases that will be created are [PostgreSQL](#) databases. Currently, no other databases are supported.

The process of creating a new persistent database can be summed up in the following steps:

1. register a new persistent store in the *app configuration file*,
2. create a data model to define the table structure of the database,
3. write a persistent store initialization function, and
4. use the Tethys command line interface to create the persistent store.

More detailed descriptions of each step of the persistent store process will be discussed in this article.

Persistent Store Registration

Registering new *persistent stores* is accomplished by adding the `persistent_stores()` method to your *app class*, which is located in your *app configuration file* (`app.py`). This method must return a list or tuple of `PersistentStore` objects. The following example illustrates what an *app class* with the `persistent_stores()` method would look like:

```

from tethys_apps.base import TethysAppBase, url_map_maker, PersistentStore

class MyFirstApp(TethysAppBase):
    """
    Tethys App Class for My First App.
    """

    ...

    def persistent_stores(self):
        """
        Add one or more persistent stores
        """
        stores = (PersistentStore(name='example_db',
                                   initializer='init_stores:init_example_db'
                                   ),
                 )

        return stores

```

Caution: The ellipsis in the code block above indicates code that is not shown for brevity. **DO NOT COPY VERBATIM.**

In this example, a database called “example_db” would be created for this app. It would be initialized by a function called “init_example_db”, which is located in a Python module called `init_stores.py`. Notice that the path to the initializer function is given using dot notation with a colon delineating the function (e.g.: `'foo.bar:function'`).

Up to 5 databases can be created for an app using the Persistent Store API. Databases follow a specific naming convention that is essentially a combination of the app name and the name that is provided during registration. For example, the database for the example above may have a name “my_first_app_example_db”. To register another database, add another Persistent Store object to the tuple that is returned by the `persistent_stores()` method.

Data Model Definition

The tables for a persistent store should be defined using an SQLAlchemy data model. The recommended location for data model code is `model.py` file that is generated with the scaffold. If your data model requires multiple files, it is recommended that you replace the `model.py` module with a package called `model` and store all of the model related modules in this package. The following example illustrates what a typical SQLAlchemy data model may consist of:

```

from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, Float
from sqlalchemy.orm import sessionmaker

from .utilities import get_persistent_store_engine

# DB Engine, sessionmaker, and base
engine = get_persistent_store_engine('example_db')
SessionMaker = sessionmaker(bind=engine)
Base = declarative_base()

# SQLAlchemy ORM definition for the stream_gages table
class StreamGage(Base):
    """
    Example SQLAlchemy DB Model

```

```
'''
__tablename__ = 'stream_gages'

# Columns
id = Column(Integer, primary_key=True)
latitude = Column(Float)
longitude = Column(Float)
value = Column(Integer)

def __init__(self, latitude, longitude, value):
    """
    Constructor for a gage
    """
    self.latitude = latitude
    self.longitude = longitude
    self.value = value
```

Object Relational Mapping

Each class in an SQLAlchemy data model defines a table in the database. Each object instantiated using an SQLAlchemy class represents individual rows or records in the table. The entire contents of a table could be represented as a list of SQLAlchemy objects. This pattern for interacting between database tables using objects in code is called Object Relational Mapping or ORM.

The example above consists of a single table called “stream_gages”, as denoted by the `__tablename__` property of the `StreamGage` class. The `StreamGage` class is defined as an SQLAlchemy data model class because it inherits from the `Base` class that was created in the previous lines using the `declarative_base()` function provided by SQLAlchemy. This inheritance notifies SQLAlchemy that the `StreamGage` class is part of the data model. All tables belonging to the same data model should inherit from the same `Base` class.

The columns of tables defined using SQLAlchemy classes are defined by properties that contain `Column` objects. The class in the example above defines four columns for the “stream_gages” table: `id`, `latitude`, `longitude`, and `value`. The column type and options are defined by the arguments passed to the `Column` constructor. For example, the `latitude` column is of type `Float` while the `id` column is of type `Integer` and is also flagged as the primary key for the table.

Engine Object

Anytime you wish to query a persistent store database, you will need to connect to it. In SQLAlchemy, the connection to a database is provided via an `engine` objects. You can retrieve the SQLAlchemy `engine` object for a persistent store database using the `get_persistent_store_engine()` function provided by the Persistent Store API. The example above shows how the `get_persistent_store_engine()` function should be used. Provide the name of the persistent store to the function and it will return the `engine` object for that store.

Note: Although the full name of the persistent store database follows the app-database naming convention described in [Persistent Store Registration](#), you need only use the name you provided during registration to retrieve the engine using `get_persistent_store_engine()`.

Session Object

Database queries are issued using SQLAlchemy `session` objects. You need to create new session objects each time you perform a new set of queries (i.e.: in each controller). Creating `session` objects is done via a

`SessionMaker`. In the example above, the `SessionMaker` is created using the `sessionmaker()` function provided by SQLAlchemy. The `SessionMaker` is bound to the `engine` object. This means that anytime a `session` is created using that `SessionMaker` it will automatically be connected to the database that the `engine` provides a connection to. You should create a `SessionMaker` for each persistent store that you create. An example of how to use `session` and `SessionMaker` objects is shown in the [Initialization Function](#) section.

SQLAlchemy ORM is a powerful tool for working with SQL databases. As a primer to SQLAlchemy ORM, we highly recommend you complete the [Object Relational Tutorial](#).

Initialization Function

The code for initializing a persistent store database should be defined in an initialization function. The recommended location for initialization functions is the `:file:init_stores.py` file that is generated with the scaffold. In most cases, each persistent store should have it's own initialization function. The initialization function makes use of the SQLAlchemy data model to create the tables and load any initial data the database may need. The following example illustrates a typical initialization function for a persistent store database:

```
from .model import engine, SessionMaker, Base, StreamGage

def init_example_db(first_time):
    """
    An example persistent store initializer function
    """
    # Create tables
    Base.metadata.create_all(engine)

    # Initial data
    if first_time:
        # Make session
        session = SessionMaker()

        # Gage 1
        gage1 = StreamGage(latitude=40.23812952992122,
                           longitude=-111.69585227966309,
                           value=1)

        session.add(gage1)

        # Gage 2
        gage2 = StreamGage(latitude=40.238784729316215,
                           longitude=-111.7101001739502,
                           value=2)

        session.add(gage2)

    session.commit()
```

Create Tables

The SQLAlchemy `Base` class defined in the data model is used to create the tables. Every class that inherits from the `Base` class is tracked by a `metadata` object. As the name implies, the `metadata` object collects metadata about each table defined by the classes in the data model. This information is used to create the tables when the `metadata.create_all()` method is called. In other words, the tables for persistent stores are created using a single line of code:

```
Base.metadata.create_all(engine)
```

Note: The `metadata.create_all()` method accepts the `engine` object for connection information.

Initial Data

The initialization functions should also be used to add any initial data to persistent store databases. The `first_time` parameter is provided to all initialization functions as an aid to adding initial data. It is a boolean that is `True` if the function is being called after the tables have been created for the first time. This is provided as a mechanism for adding initial data only the first time the initialization function is run. Notice the code that adds initial data to the persistent store database in the example above is wrapped in a conditional statement that uses the `first_time` parameter.

Example SQLAlchemy Query

This initial data code uses an SQLAlchemy data model to add four stream gages to the persistent store database. A new `session` object is created using the `SessionMaker` that was defined in the model. Creating a new record in the database using SQLAlchemy is achieved by creating a new `StreamGage` object and adding it to the `session` object using the `session.add()` method. The `session.commit()` method is called, to persist the new records to the persistent store database.

Managing Persistent Stores

Persistent store management is handled via the `syncstores` command provided by the Tethys Command Line Interface (Tethys CLI). This command is used to create the persistent stores of apps during installation. It should also be used anytime you make changes to persistent store registration, data models, or initialization functions. For example, after performing the registration, creating the data model, and defining the initialization function in the example above, the `syncstores` command would need to be called from the command line to create the new persistent store:

```
$ tethys syncstores my_first_app
```

This command would create all the non-existent persistent stores that are registered for `my_first_app` and run the initialization functions for them. This is the most basic usage of the `syncstores` command. A detailed description of the `syncstores` command can be found in the [Command Line Interface](#) documentation.

API Documentation

See [App Base Class API](#) for an explanation of the `TethysAppBase.persistent_stores()` method.

Note: Tethys app projects generated using the scaffold include a function in the `utilities.py` called `get_persistent_store_engine()`. This function is a wrapper for the function defined above and it allows the omission of the `app_name` parameter for simplicity. The example above uses the `utilities.get_persistent_store_engine()` function.

1.6.6 Spatial Persistent Stores API

Last Updated: November 24, 2014

Persistent store databases can support spatial data types. The spatial capabilities are provided by the [PostGIS](#) extension for the [PostgreSQL](#) database. PostGIS extends the column types of PostgreSQL databases by adding `geometry`,

geography, and raster types. PostGIS also provides hundreds of database functions that can be used to perform spatial operations on data stored in spatial columns. For more information on PostGIS, see <http://www.postgis.net>.

The following article details the the spatial capabilities of persistent stores in Tethys Platform. This article builds on the concepts and ideas introduced in the *Persistent Stores API* documentation. Please review it before continuing.

Register Spatial Persistent Store

Registering spatially enabled persistent stores follows the same process as registering normal persistent stores. The only difference is that you will set the `spatial` attribute of the `PersistentStore` object to `True`:

```
from tethys_apps.base import TethysAppBase, url_map_maker, PersistentStore

class MyFirstApp(TethysAppBase):
    """
    Tethys App Class for My First App.
    """
    ...

    def persistent_stores(self):
        """
        Add one or more persistent stores
        """
        stores = (PersistentStore(name='spatial_db',
                                initializer='init_stores:init_spatial_db',
                                spatial=True
                                ),
                )

        return stores
```

Caution: The ellipsis in the code block above indicates code that is not shown for brevity. **DO NOT COPY VERBATIM.**

Adding Spatial Columns to Model

Working with the `raster`, `geometry`, and `geography` column types provided by PostGIS is not supported natively in SQLAlchemy. For this, Tethys Platform provides the `GeoAlchemy2`, which extends SQLAlchemy to support spatial columns and database functions. A data model that uses a `geometry` column type to store the points for stream gages may look like this:

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer
from sqlalchemy.orm import sessionmaker

from geoalchemy2 import Geometry

from .utilities import get_persistent_store_engine

# Spatial DB Engine, sessionmaker, and base
spatial_engine = get_persistent_store_engine('spatial_db')
SpatialSessionMaker = sessionmaker(bind=spatial_engine)
SpatialBase = declarative_base()
```



```
# SQLAlchemy ORM definition for the spatial_stream_gages table
class SpatialStreamGage(SpatialBase):
    """
    Example of SQLAlchemy spatial DB model
    """
    __tablename__ = 'spatial_stream_gages'

    # Columns
    id = Column(Integer, primary_key=True)
    value = Column(Integer)
    geom = Column(Geometry('POINT'))

    def __init__(self, latitude, longitude, value):
        """
        Constructor for a gage
        """
        self.geom = 'SRID=4326;POINT({0} {1})'.format(longitude, latitude)
        self.value = value
```

This data model is very similar to the data model defined in the *Persistent Stores API* documentation. Rather than using Float columns to store the latitude and longitude coordinates, the spatial data model uses a GeoAlchemy2 Geometry column called “geom”. Notice that the constructor (`__init__.py`) takes the latitude and longitude provided and sets the value of the geom column to a string with a special format called [Well Known Text](#). This is a common pattern when working with GeoAlchemy2 columns.

Important: This article only briefly introduces the concepts of working with GeoAlchemy2. It is highly recommended that you complete the [GeoAlchemy ORM](#) tutorial.

Initialization Function

Initializing spatial persistent stores is performed in exactly the same way as normal persistent stores. An initialization function for the example above, would look like this:

```
from .model import spatial_engine, SpatialSessionMaker, SpatialBase, SpatialStreamGage

def init_spatial_db(first_time):
    """
    An example persistent store initializer function
    """
    # Create tables
    SpatialBase.metadata.create_all(spatial_engine)

    # Initial data
    if first_time:
        # Make session
        session = SpatialSessionMaker()

        # Gage 1
        gage1 = SpatialStreamGage(latitude=40.23812952992122,
                                  longitude=-111.69585227966309,
                                  value=1)

        session.add(gage1)

    # Gage 2
```

```
gage2 = SpatialStreamGage(latitude=40.238784729316215,
                          longitude=-111.7101001739502,
                          value=2)

session.add(gage2)

session.commit()
```

Using Spatial Database Functions

One of the major advantages of storing spatial data in PostGIS is that the data is exposed to spatial querying. PostGIS includes over 400 database functions (not counting variants) that can be used to perform spatial operations on the data stored in the database. Refer to the [Geometry Function Reference](#) and the [Raster Function Reference](#) in the PostGIS documentation for more details.

GeoAlchemy2 makes it easy to use the spatial functions provided by PostGIS to perform spatial queries. For example, the `ST_Contains` function can be used to determine if one geometry is contained inside another geometry. To perform this operation on the spatial stream gage model would look something like this:

```
from sqlalchemy import func
from .model import SpatialStreamGage, SpatialSessionMaker

session = SpatialSessionMaker()
query = session.query(SpatialStreamGage).filter(
    func.ST_Contains('POLYGON((0 0,0 1,1 1,0 1,0 0))', SpatialStreamGage.geom)
)
```

Important: This article only briefly introduces the concepts of working with GeoAlchemy2. It is highly recommended that you complete the [GeoAlchemy ORM tutorial](#).

1.6.7 Dataset Services API

Last Updated: May 13, 2015

Dataset services are web services external to Tethys Platform that can be used to store and publish file-based *datasets* (e.g.: text files, Excel files, zip archives, other model files). Tethys app developers can use the Dataset Services API to access *datasets* for use in their apps and publish any resulting *datasets* their apps may produce. [CKAN](#) is currently the only supported dataset service.

Key Concepts

Tethys Dataset Services API provides a standardized interface for interacting with *dataset services*. This means that you can use datasets from different sources without completely overhauling your code. Each of the supported *dataset services* provides a `DatasetEngine` object with the same methods. For example, all `DatasetEngine` objects have a method called `list_datasets()` that will have the same result, returning a list of the datasets that are available.

There are two important definitions that are applicable to *dataset services*: *dataset* and *resource*. A *resource* contains a single file or other object and the metadata associated with it. A *dataset* is a container for one or more resources.

Dataset Service Engine References

All `DatasetEngine` objects implement a minimum set of base methods. However, some `DatasetEngine` objects may include additional methods that are unique to that `DatasetEngine` and the arguments that each method accepts may vary slightly. Refer to the following references for the methods that are offered by each `DatasetEngine`.

Base Dataset Engine Reference

Last Updated: January 19, 2015

All `DatasetEngine` object provide a minimum set of methods for interacting with *datasets* and *resources*. Specifically, the methods allow the standard CRUD operations (Create, Read, Update, Delete) for both *datasets* and *resources*.

All `DatasetEngine` methods return a dictionary, often called the Response dictionary. The Response dictionary contains an item named 'success' that contains a boolean indicating whether the operation was successful or not. If 'success' is `True`, then the the dictionary will also have an item named 'result' that contains the result of the operation. If 'success' is `False`, then the Response dictionary will contain an item called 'error' with information about what went wrong.

The following reference provides a summary of the base methods and properties provided by all `DatasetEngine` objects.

Properties `DatasetEngine`. **endpoint** (string): URL for the *dataset service* API endpoint.

`DatasetEngine`. **apikey** (string, optional): API key may be used for authorization.

`DatasetEngine`. **username** (string, optional): Username key may be used for authorization.

`DatasetEngine`. **password** (string, optional): Password key may be used for authorization.

`DatasetEngine`. **type** (string, readonly): Identifies the type of `DatasetEngine` object.

Create Methods

`DatasetEngine`. **create_dataset** (*name*, ****kwargs**)

Create a new dataset.

Parameters

- **name** (*string*) – Name of the dataset to create.
- ****kwargs** (*kwargs*, *optional*) – Any number of additional keyword arguments.

Returns Response dictionary

Return type (dict)

`DatasetEngine`. **create_resource** (*dataset_id*, *url=None*, *file=None*, ****kwargs**)

Create a new resource.

Parameters

- **dataset_id** (*string*) – Identifier of the dataset to which the resource will be added.
- **url** (*string*, *optional*) – URL of resource to associate with resource.
- **file** (*string*, *optional*) – Path of file to upload as resource.
- ****kwargs** (*kwargs*, *optional*) – Any number of additional keyword arguments.

Returns Response dictionary

Return type (dict)

Read Methods

`DatasetEngine.get_dataset(dataset_id, **kwargs)`

Retrieve a dataset object.

Parameters

- **dataset_id** (*string*) – Identifier of the dataset to retrieve.
- ****kwargs** (*kwargs, optional*) – Any number of additional keyword arguments.

Returns Response dictionary

Return type (dict)

`DatasetEngine.get_resource(resource_id, **kwargs)`

Retrieve a resource object.

Parameters

- **resource_id** (*string*) – Identifier of the dataset to retrieve.
- ****kwargs** (*kwargs, optional*) – Any number of additional keyword arguments.

Returns Response dictionary

Return type (dict)

`DatasetEngine.search_datasets(query, **kwargs)`

Search for datasets that match a query.

Parameters

- **query** (*dict*) – Key value pairs representing the fields and values of the datasets to be included.
- ****kwargs** – Any number of additional keyword arguments.

Returns Response dictionary

Return type (dict)

`DatasetEngine.search_resources(query, **kwargs)`

Search for resources that match a query.

Parameters

- **query** (*dict*) – Key value pairs representing the fields and values of the resources to be included.
- ****kwargs** (*kwargs, optional*) – Any number of additional keyword arguments.

Returns Response dictionary

Return type (dict)

`DatasetEngine.list_datasets(**kwargs)`

List all datasets available from the dataset service.

Parameters ****kwargs** (*kwargs, optional*) – Any number of additional keyword arguments.

Returns Response dictionary

Return type (dict)

Update Methods

`DatasetEngine.update_dataset(dataset_id, **kwargs)`

Update an existing dataset.

Parameters

- **dataset_id** (*string*) – Identifier of the dataset to update.
- ****kwargs** (*kwargs, optional*) – Any number of additional keyword arguments.

Returns Response dictionary

Return type (dict)

`DatasetEngine.update_resource(resource_id, url=None, file=None, **kwargs)`

Update an existing resource.

Parameters

- **resource_id** (*string*) – Identifier of the resource to update.
- **url** (*string*) – URL of resource to associate with resource.
- **file** (*string*) – Path of file to upload as resource.
- ****kwargs** (*kwargs, optional*) – Any number of additional keyword arguments.

Returns Response dictionary

Return type (dict)

Delete Methods

`DatasetEngine.delete_dataset(dataset_id, **kwargs)`

Delete a dataset.

Parameters

- **dataset_id** (*string*) – Identifier of the dataset to delete.
- ****kwargs** (*kwargs, optional*) – Any number of additional keyword arguments.

Returns Response dictionary

Return type (dict)

`DatasetEngine.delete_resource(resource_id, **kwargs)`

Delete a resource.

Parameters

- **resource_id** (*string*) – Identifier of the resource to delete.
- ****kwargs** (*kwargs, optional*) – Any number of additional keyword arguments.

Returns Response dictionary

Return type (dict)

CKAN Dataset Engine Reference

Last Updated: January 19, 2015

The following reference provides a summary the class used to define the `CkanDatasetEngine` objects.

```
class tethys_dataset_services.engines.CkanDatasetEngine (endpoint, apikey=None,
                                                         username=None, password=None)
```

Definition for CKAN Dataset Engine objects.

```
create_dataset (name, console=False, **kwargs)
```

Create a new CKAN dataset.

Wrapper for the CKAN `package_create` API method. See the CKAN API docs for this method to see applicable options (<http://docs.ckan.org/en/ckan-2.2/api.html>).

Parameters

- **name** (*string*) – The id or name of the resource to retrieve.
- **console** (*bool, optional*) – Pretty print the result to the console for debugging. Defaults to False.
- ****kwargs** – Any number of optional keyword arguments for the method (see CKAN docs).

Returns The response dictionary or None if an error occurs.

```
create_resource (dataset_id, url=None, file=None, console=False, **kwargs)
```

Create a new CKAN resource.

Wrapper for the CKAN `resource_create` API method. See the CKAN API docs for this method to see applicable options (<http://docs.ckan.org/en/ckan-2.2/api.html>).

Parameters

- **dataset_id** (*string*) – The id or name of the dataset to to which the resource will be added.
- **url** (*string, optional*) – URL for the resource that will be added to the dataset.
- **file** (*string, optional*) – Absolute path to a file to upload for the resource.
- **console** (*bool, optional*) – Pretty print the result to the console for debugging. Defaults to False.
- ****kwargs** – Any number of optional keyword arguments for the method (see CKAN docs).

Returns The response dictionary or None if an error occurs.

```
delete_dataset (dataset_id, console=False, **kwargs)
```

Delete CKAN dataset

Wrapper for the CKAN `package_delete` API method. See the CKAN API docs for this method to see applicable options (<http://docs.ckan.org/en/ckan-2.2/api.html>).

Parameters

- **dataset_id** (*string*) – The id or name of the dataset to delete.
- **console** (*bool, optional*) – Pretty print the result to the console for debugging. Defaults to False.
- ****kwargs** – Any number of optional keyword arguments for the method (see CKAN docs).

Returns The response dictionary or None if an error occurs.

```
delete_resource (resource_id, console=False, **kwargs)
```

Delete CKAN resource

Wrapper for the CKAN `resource_delete` API method. See the CKAN API docs for this method to see applicable options (<http://docs.ckan.org/en/ckan-2.2/api.html>).

Parameters

- **resource_id** (*string*) – The id of the resource to delete.
- **console** (*bool, optional*) – Pretty print the result to the console for debugging. Defaults to False.
- ****kwargs** – Any number of optional keyword arguments for the method (see CKAN docs).

Returns The response dictionary or None if an error occurs.

get_dataset (*dataset_id, console=False, **kwargs*)

Retrieve CKAN dataset

Wrapper for the CKAN `package_show` API method. See the CKAN API docs for this method to see applicable options (<http://docs.ckan.org/en/ckan-2.2/api.html>).

Parameters

- **dataset_id** (*string*) – The id or name of the dataset to retrieve.
- **console** (*bool, optional*) – Pretty print the result to the console for debugging. Defaults to False.
- ****kwargs** – Any number of optional keyword arguments for the method (see CKAN docs).

Returns The response dictionary or None if an error occurs.

get_resource (*resource_id, console=False, **kwargs*)

Retrieve CKAN resource

Wrapper for the CKAN `resource_show` API method. See the CKAN API docs for this method to see applicable options (<http://docs.ckan.org/en/ckan-2.2/api.html>).

Parameters

- **resource_id** (*string*) – The id of the resource to retrieve.
- **console** (*bool, optional*) – Pretty print the result to the console for debugging. Defaults to False.
- ****kwargs** – Any number of optional keyword arguments for the method (see CKAN docs).

Returns The response dictionary or None if an error occurs.

list_datasets (*with_resources=False, console=False, **kwargs*)

List CKAN datasets.

Wrapper for the CKAN `package_list` and `current_package_list_with_resources` API methods. See the CKAN API docs for these methods to see applicable options (<http://docs.ckan.org/en/ckan-2.2/api.html>).

Parameters

- **with_resources** (*bool, optional*) – Return a list of dataset dictionaries. Defaults to False.
- **console** (*bool, optional*) – Pretty print the result to the console for debugging. Defaults to False.
- ****kwargs** – Any number of optional keyword arguments for the method (see CKAN docs).

Returns A list of dataset names or a list of dataset dictionaries if `with_resources` is true.

Return type list

search_datasets (*query, console=False, **kwargs*)

Search CKAN datasets that match a query.

Wrapper for the CKAN `search_datasets` API method. See the CKAN API docs for this methods to see applicable options (<http://docs.ckan.org/en/ckan-2.2/api.html>).

Parameters

- **query** (*dict*) – Key value pairs representing field and values to search for.
- **console** (*bool, optional*) – Pretty print the result to the console for debugging. Defaults to False.
- ****kwargs** – Any number of optional keyword arguments for the method (see CKAN docs).

Returns The response dictionary or None if an error occurs.

search_resources (*query, console=False, **kwargs*)

Search CKAN resources that match a query.

Wrapper for the CKAN search_resources API method. See the CKAN API docs for this methods to see applicable options (<http://docs.ckan.org/en/ckan-2.2/api.html>).

Parameters

- **query** (*dict*) – Key value pairs representing field and values to search for.
- **console** (*bool, optional*) – Pretty print the result to the console for debugging. Defaults to False.
- ****kwargs** – Any number of optional keyword arguments for the method (see CKAN docs).

Returns The response dictionary or None if an error occurs.

type

CKAN Dataset Engine Type

update_dataset (*dataset_id, console=False, **kwargs*)

Update CKAN dataset

Wrapper for the CKAN package_update API method. See the CKAN API docs for this method to see applicable options (<http://docs.ckan.org/en/ckan-2.2/api.html>).

Parameters

- **dataset_id** (*string*) – The id or name of the dataset to update.
- **console** (*bool, optional*) – Pretty print the result to the console for debugging. Defaults to False.
- ****kwargs** – Any number of optional keyword arguments for the method (see CKAN docs).

Returns The response dictionary or None if an error occurs.

update_resource (*resource_id, url=None, file=None, console=False, **kwargs*)

Update CKAN resource

Wrapper for the CKAN resource_update API method. See the CKAN API docs for this method to see applicable options (<http://docs.ckan.org/en/ckan-2.2/api.html>).

Parameters

- **resource_id** (*string*) – The id of the resource that will be updated.
- **url** (*string, optional*) – URL of the resource that will be added to the dataset.
- **file** (*string, optional*) – Absolute path to a file to upload for the resource.
- **console** (*bool, optional*) – Pretty print the result to the console for debugging. Defaults to False.
- ****kwargs** – Any number of optional keyword arguments for the method (see CKAN docs).

Returns The response dictionary or None if an error occurs.

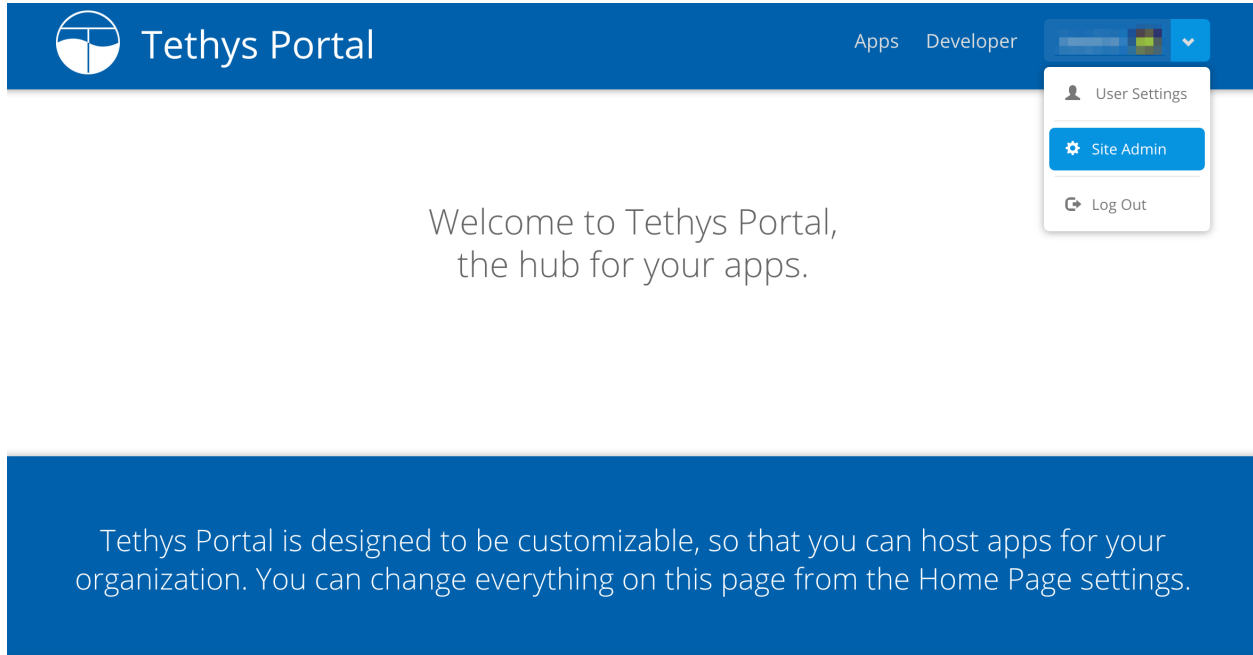
`validate()`

Validate CKAN dataset engine. Will throw an error if not valid.

Register New Dataset Service

Registering new dataset services is performed through the System Admin Settings.


1. Login to your Tethys Platform instance as an administrator.
2. Select “Site Admin” from the user drop down menu.



3. Select “Dataset Services” from the “Tethys Services” section.
4. Select an existing Dataset Service configuration from the list to edit it OR click on the “Add Dataset Service” button to create a new one.
5. Give the Dataset Service configuration a name, select an appropriate engine, and specify the endpoint. The name must be unique, because it is used to retrieve the Dataset Service connection object. The endpoint is a URL pointing to the Dataset Service API. Example endpoints for several different types of Dataset Services are shown below:

```
# CKAN Endpoint URL  
http://www.example.com/api/3/action
```

If authentication is required, specify either the API key or the username and password.

 Tethys Portal
Apps Developer


Site Administration


Authentication and Authorization	
Groups	Add Change
Users	Add Change
Tethys Compute	
Clusters	Add Change
Settings	Change
Tethys Portal	
Site Settings	Change
Tethys Services	
Dataset Services	Add Change
Spatial Dataset Services	Add Change
Web Processing Services	Add Change

Recent Actions

My Actions

- [General Settings](#)
Settings Category
- [default](#)
Web Processing Service
- [default_geoserver](#)
Spatial Dataset Service
- [default](#)
Web Processing Service
- [default_geoserver](#)
Spatial Dataset Service
- [default_geoserver](#)
Spatial Dataset Service
- [default](#)
Dataset Service
- [default](#)
Dataset Service
- [default_ckan](#)
Dataset Service
- [default](#)
Dataset Service

Copyright © 2015 Your Organization
Powered by  Tethys Platform

 Tethys Portal
Apps Developer

[Home](#) > [Tethys Services](#) > [Dataset Services](#)


Select Dataset Service To Change

[Add Dataset Service +](#)

Action: [Go](#) 0 of 2 selected

<input type="checkbox"/>	Dataset Service
<input type="checkbox"/>	default_ckan
<input type="checkbox"/>	default

2 Dataset Services

Copyright © 2015 Your Organization
Powered by  Tethys Platform

Note: When linking Tethys to a CKAN dataset service, an API Key is required. All user accounts are issued an API key. To access the API Key log into the CKAN on which you have an account and browse to your user profile page. The API key will be listed there. Depending on the CKAN instance and the dataset, you may have full read-write access or you may have read-only access.

When you are done, the form should look similar to this:

The screenshot shows the Tethys Portal interface. At the top, there is a blue header with the Tethys Portal logo and navigation links for 'Apps' and 'Developer'. Below the header, a breadcrumb trail reads 'Home > Tethys Services > Dataset Services > default_ckan'. The main content area is titled 'Change Dataset Service' and includes a 'History' button. The form contains several input fields: 'Name' (default_ckan), 'Engine' (CKAN), 'Endpoint' (http://ciwckan.chpc.utah.edu), 'Apikey' (tHIS-is-mY-@PI-k3Y), 'Username', and 'Password'. At the bottom of the form, there are three buttons: 'Delete' (with a red asterisk), 'Save and add another', and 'Save and continue editing' (with a green 'Save' button). The footer of the page contains copyright information for 'Your Organization' and a 'Powered by Tethys Platform' logo.

6. Press “Save” to save the Dataset Service configuration.

Note: Prior to version Tethys Platform 1.1.0, it was possible to register dataset services using a mechanism in the *app configuration file*. This mechanism has been deprecated due to security concerns.

Working with Dataset Services

After dataset services have been properly configured, you can use the services to store and retrieve data for your apps. The process involves the following steps:

1. Get a Dataset Service Engine

The Dataset Services API provides a convenience function for working with *dataset services* called `get_dataset_engine`. To retrieve and engine for a sitewide configuration, call `get_dataset_engine` with the name of the configuration:

```
from tethys_apps.sdk import get_dataset_engine

dataset_engine = get_dataset_engine(name='example')
```

It will return the first service with a matching name or raise an exception if the service cannot be found with the given name. Alternatively, you may retrieve a list of all the dataset engine objects that are registered using the `list_dataset_engines` function:

```
from tethys_apps.sdk import list_dataset_engines

dataset_engines = list_dataset_engines()
```

You can also create a `DatasetEngine` object directly without using the convenience function. This can be useful if you want to vary the credentials for dataset access frequently (e.g.: using user specific credentials). Simply import it and instantiate it with valid credentials:

```
from tethys_dataset_services.engines import CkanDatasetEngine

dataset_engine = CkanDatasetEngine(endpoint='http://www.example.com/api/3/action',
apikey='a-R311Y-n1Ce-@Pi-key')
```

Caution: Take care not to store API keys, usernames, or passwords in the source files of your app—especially if the source is made public. This could compromise the security of the dataset service.

2. Use the Dataset Service Engine

After you have a `DatasetEngine`, simply call the desired method on it. All `DatasetEngine` methods return a dictionary with an item named `'success'` that contains a boolean. If the operation was successful, the value of `'success'` will be `True`, otherwise it will be `False`. If the value of `'success'` is `True`, the dictionary will also contain an item named `'result'` that will contain the results. If it is `False`, the dictionary will contain an item named `'error'` that will contain information about the error that occurred. This can be used for debugging purposes as illustrated in the following example:

```
from tethys_apps.sdk import get_dataset_engine

dataset_engine = get_dataset_engine(name='example')

result = dataset_engine.list_datasets()

if result['success']:
    dataset_list = result['result']

    for each dataset in dataset_list:
        print dataset
else:
    print(result['error'])
```

Use the dataset service engines references above for descriptions of the methods available and examples.

1.6.8 Spatial Dataset Services API

Last Updated: May 13, 2015

Spatial dataset services are web services that can be used to store and publish file-based *spatial datasets* (e.g.: Shapefile and GeoTiff). The spatial datasets published using spatial dataset services are made available in a variety of formats, many of which or more web friendly than the native format (e.g.: PNG, JPEG, GeoJSON, and KML). Tethys app

developers can use this Spatial Dataset Services API to store and access :term:` spatial datasets` for use in their apps and publish any resulting *datasets* their apps may produce.

Powered by GeoServer

GeoServer powers the Spatial Dataset Service capabilities of Tethys Platform. It is capable of storing and serving vector and raster datasets in several popular formats including Shapefiles, GeoTiff, ArcGrid and others. GeoServer serves the data in a variety of formats via the [Open Geospatial Consortium \(OGC\)](#) standards including [Web Feature Service \(WFS\)](#), [Web Map Service \(WMS\)](#), and [Web Coverage Service \(WCS\)](#).

Key Concepts

There are quite a few concepts that you should understand before working with GeoServer and spatial dataset services. Definitions of each are provided here for quick reference.

Resources are the spatial datasets. These can vary in format ranging from a single file or multiple files to database tables depending on the type resource.

Feature Type: is a type of *resource* containing vector data or data consisting of discreet features such as points, lines, or polygons and any tables of attributes that describe the features.

Coverage: is a type of *resource* containing raster data or numeric gridded data.

Layers: are *resources* that have been published. Layers associate styles and other settings with the *resource* that are needed to generate maps of the *resource* via OGC services.

Layer Groups: are preset groups of *layers* that can be served as WMS services as though they were one *layer*.

Stores: represent repositories of spatial datasets such as database tables or directories of shapefiles. A *store* containing only *feature types* is called a **Data Store** and a *store* containing only *coverages* is called a **Coverage Store**.

Workspaces: are arbitrary groupings of data to help with organization of the data. It would be a good idea to store all of the spatial datasets for your app in a workspace resembling the name of your app to avoid conflicts with other apps.

Styles: are a set of rules that dictate how a *layer* will be rendered when accessed via WMS. A *layer* may be associated with many styles and a style may be associated with many *layers*. Styles on GeoServer are written in [Styled Layer Descriptor \(SLD\)](#) format.

Styled Layer Descriptor (SLD): An XML-based markup language that can be used to specify how spatial datasets should be rendered. See GeoServer's [SLD Cookbook](#) for a good primer on SLD.

Web Feature Service (WFS): An OGC standard for exchanging vector data (i.e.: feature types) over the internet. WFS can be used to not only query for the features (points, lines, and polygons) but also the attributes associated with the features.

Web Coverage Service (WCS): An OGC standard for exchanging raster data (i.e.: coverages) over the internet. WCS is roughly the equivalent of WFS but for *coverages*, access to the raw coverage information, not just the image.

Web Mapping Service (WMS): An OGC standard for generating and exchanging maps of spatial data over the internet. WMS can be used to compose maps of several different spatial dataset sources and formats.

Spatial Dataset Engine References

All `SpatialDatasetEngine` objects implement a minimum set of base methods. However, some `SpatialDatasetEngine` objects may include additional methods that are unique to that `SpatialDatasetEngine` implementation and the arguments that each method accepts may vary slightly. Refer to the following references for the methods that are offered by each `SpatialDatasetEngine`.

Base Spatial Dataset Engine Reference

Last Updated: January 30, 2015

All `SpatialDatasetEngine` objects provide a minimum set of methods for interacting with layers and resources. Specifically, the methods allow the standard CRUD operations (Create, Read, Update, Delete) for both layers and resources.

All `SpatialSpatialDatasetEngine` methods return a dictionary called the response dictionary. The Response dictionary contains an item named 'success' that is a boolean indicating whether the operation was successful or not. If 'success' is True, then the the dictionary will also have an item named 'result' that contains the result of the operation. If 'success' is False, then the Response dictionary will contain an item called 'error' with information about what went wrong.

The following reference provides a summary of the base methods and properties provided by all `SpatialDatasetEngine` objects.

Properties `SpatialDatasetEngine`. **endpoint** (string): URL for the spatial dataset service API endpoint.

`SpatialDatasetEngine`. **apikey** (string, optional): API key may be used for authorization.

`SpatialDatasetEngine`. **username** (string, optional): Username key may be used for authorization.

`SpatialDatasetEngine`. **password** (string, optional): Password key may be used for authorization.

`SpatialDatasetEngine`. **type** (string, readonly): Identifies the type of `SpatialDatasetEngine` object.

Create Methods

`SpatialDatasetEngine`. **create_resource** (*resource_id*, ***kwargs*)

Create a new resource.

Parameters

- **resource_id** (*string*) – Identifier of the resource to create.
- ****kwargs** (*kwargs*, *optional*) – Any number of additional keyword arguments.

Returns Response dictionary

Return type (dict)

`SpatialDatasetEngine`. **create_layer** (*layer_id*)

Create a new layer.

Parameters **layer_id** (*string*) – Identifier of the layer to create.

Returns Response dictionary

Return type (dict)

Read Methods

`SpatialDatasetEngine`. **get_resource** (*resource_id*)

Retrieve a resource object.

Parameters **resource_id** (*string*) – Identifier of the dataset to retrieve.

Returns Response dictionary

Return type (dict)

`SpatialDatasetEngine`. **get_layer** (*layer_id*)

Retrieve a single layer object.

Parameters `layer_id` (*string*) – Identifier of the layer to retrieve.

Returns Response dictionary

Return type (dict)

`SpatialDatasetEngine.list_resources()`

List all resources available from the spatial dataset service.

Returns Response dictionary

Return type (dict)

`SpatialDatasetEngine.list_layers()`

List all layers available from the spatial dataset service.

Returns Response dictionary

Return type (dict)

Update Methods

`SpatialDatasetEngine.update_resource(resource_id, **kwargs)`

Update an existing resource.

Parameters

- **resource_id** (*string*) – Identifier of the resource to update.
- ****kwargs** (*kwargs, optional*) – Any number of additional keyword arguments.

Returns Response dictionary

Return type (dict)

`SpatialDatasetEngine.update_layer(layer_id, **kwargs)`

Update an existing layer.

Parameters

- **layer_id** (*string*) – Identifier of the layer to update.
- ****kwargs** (*kwargs, optional*) – Any number of additional keyword arguments.

Returns Response dictionary

Return type (dict)

Delete Methods

`SpatialDatasetEngine.delete_resource(resource_id)`

Delete a resource.

Parameters **resource_id** (*string*) – Identifier of the resource to delete.

Returns Response dictionary

Return type (dict)

`SpatialDatasetEngine.delete_layer(layer_id)`

Delete a layer.

Parameters **layer_id** (*string*) – Identifier of the layer to delete.

Returns Response dictionary

Return type (dict)

GeoServer Spatial Dataset Engine Reference

Last Updated: January 30, 2015

The following reference provides a summary the class used to define the `GeoServerSpatialDatasetEngine` objects.

```
class tethys_dataset_services.engines.GeoServerSpatialDatasetEngine (endpoint,
                                                                    apikey=None,
                                                                    user-
                                                                    name=None,
                                                                    pass-
                                                                    word=None)
```

Definition for GeoServer Dataset Engine objects.

```
add_table_to_postgis_store (store_id, table, debug=True)
```

Add an existing postgis table as a feature resource to a postgis store that already exists.

Parameters

- **store_id** (*string*) – Identifier for the store to add the resource to. Can be a store name or a workspace name combination (e.g.: “name” or “workspace:name”). Note that the workspace must be an existing workspace. If no workspace is given, the default workspace will be assigned.
- **table** (*string*) – Name of existing table to add as a feature resource. A layer will automatically be created for this resource. Both the resource and the layer will share the same name as the table.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.add_table_to_postgis_store(store_id='workspace:store_name', table='table_name')
```

```
create_coverage_resource (store_id, coverage_file, coverage_type, overwrite=False, de-
                           bug=False)
```

Use this method to add coverage resources to GeoServer.

This method will result in the creation of three items: a coverage store, a coverage resource, and a layer. If `store_id` references a store that does not exist, it will be created. The coverage resource and the subsequent layer will be created with the same name as the image file that is uploaded.

Parameters

- **store_id** (*string*) – Identifier for the store to add the image to or to be created. Can be a name or a workspace name combination (e.g.: “name” or “workspace:name”). Note that the workspace must be an existing workspace. If no workspace is given, the default workspace will be assigned.
- **coverage_file** (*string*) – Path to the coverage image or zip archive. Most files will require a .prj file with the Well Known Text definition of the projection. Zip this file up with the image and send the archive.
- **coverage_type** – Type of coverage that is being created. Valid values include: ‘geotiff’, ‘worldimage’, ‘imagemosaic’, ‘gtopo30’, ‘arcgrid’, and ‘grassgrid’.

- **overwrite** (*bool, optional*) – Overwrite the file if it already exists.
- **charset** (*string, optional*) – Specify the character encoding of the file being uploaded (e.g.: ISO-8559-1)
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Note: If the type coverage being uploaded includes multiple files (e.g.: image, world file, projection file), they must be uploaded as a zip archive. Otherwise upload the single file.

Returns Response dictionary

Return type (dict)

Examples

```
coverage_file = '/path/to/geotiff/example.zip'
```

```
response = engine.create_coverage_resource(store_id='workspace:store_name', coverage_file=coverage_file, coverage_type='geotiff')
```

create_layer_group (*layer_group_id, layers, styles, bounds=None, debug=False*)

Create a layer group. The number of layers and the number of styles must be the same.

Parameters

- **layer_group_id** (*string*) – Identifier of the layer group to create.
- **layers** (*iterable*) – A list of layer names to be added to the group. Must be the same length as the styles list.
- **styles** (*iterable*) – A list of style names to associate with each layer in the group. Must be the same length as the layers list.
- **bounds** (*iterable*) – A tuple representing the bounding box of the layer group (e.g.: ('-74.02722', '-73.907005', '40.684221', '40.878178', 'EPSG:4326'))
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
layers = ('layer1', 'layer2')
```

```
styles = ('style1', 'style2')
```

```
bounds = ('-74.02722', '-73.907005', '40.684221', '40.878178', 'EPSG:4326')
```

```
response = engine.create_layer_group(layer_group_id='layer_group_name', layers=layers, styles=styles, bounds=bounds)
```

create_postgis_feature_resource (*store_id, host, port, database, user, password, table=None, debug=False*)

Use this method to link an existing PostGIS database to GeoServer as a feature store. Note that this method only works for data in vector formats.

Parameters

- **store_id** (*string*) – Identifier for the store to add the resource to. Can be a store name or a workspace name combination (e.g.: “name” or “workspace:name”). Note that the workspace must be an existing workspace. If no workspace is given, the default workspace will be assigned.
- **host** (*string*) – Host of the PostGIS database (e.g.: ‘www.example.com’).
- **port** (*string*) – Port of the PostGIS database (e.g.: ‘5432’)
- **database** (*string*) – Name of the database.
- **user** (*string*) – Database user that has access to the database.
- **password** (*string*) – Password of database user.
- **table** (*string, optional*) – Name of existing table to add as a feature resource to the newly created feature store. A layer will automatically be created for the feature resource as well. Both the layer and the resource will share the same name as the table.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

With Table

```
response = engine.create_postgis_feature_resource(store_id='workspace:store_name', table='table_name', host='localhost', port='5432', database='database_name', user='user', password='pass')
```

Without table

```
response = engine.create_postgis_resource(store_id='workspace:store_name', host='localhost', port='5432', database='database_name', user='user', password='pass')
```

```
create_shapefile_resource (store_id, shapefile_base=None, shapefile_zip=None, shapefile_upload=None, overwrite=False, charset=None, debug=False)
```

Use this method to add shapefile resources to GeoServer.

This method will result in the creation of three items: a feature type store, a feature type resource, and a layer. If `store_id` references a store that does not exist, it will be created. The feature type resource and the subsequent layer will be created with the same name as the feature type store. Provide shapefile with either `shapefile_base`, `shapefile_zip`, or `shapefile_upload` arguments.

Parameters

- **store_id** (*string*) – Identifier for the store to add the resource to. Can be a store name or a workspace name combination (e.g.: “name” or “workspace:name”). Note that the workspace must be an existing workspace. If no workspace is given, the default workspace will be assigned.
- **shapefile_base** (*string, optional*) – Path to shapefile base name (e.g.: “/path/base” for shapefile at “/path/base.shp”)
- **shapefile_zip** (*string, optional*) – Path to a zip file containing the shapefile and side cars.

- **shapefile_upload** (*FileUpload list, optional*) – A list of Django FileUpload objects containing a shapefile and side cars that have been uploaded via multipart/form-data form.
- **overwrite** (*bool, optional*) – Overwrite the file if it already exists.
- **charset** (*string, optional*) – Specify the character encoding of the file being uploaded (e.g.: ISO-8559-1)
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
# For example.shp (path to file but omit the .shp extension)
shapefile_base = "/path/to/shapefile/example"

response = engine.create_shapefile_resource(store_id='workspace:store_name', shape-
file_base=shapefile_base)

# Using zip
shapefile_zip = "/path/to/shapefile/example.zip"

response = engine.create_shapefile_resource(store_id='workspace:store_name', shape-
file_zip=shapefile_zip)

# Using upload
file_list = request.FILES.getlist('files')

response = engine.create_shapefile_resource(store_id='workspace:store_name', shape-
file_upload=file_list)
```

create_style (*style_id, sld, overwrite=False, debug=False*)

Create a new SLD style object.

Parameters

- **create_style** (*string*) – Identifier of the style to create.
- **sld** (*string*) – Styled Layer Descriptor string
- **overwrite** (*bool, optional*) – Overwrite if style already exists. Defaults to False.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
sld = '/path/to/style.sld'
sld_file = open(sld, 'r')
response = engine.create_style(style_id='fred', sld=sld_file.read(), debug=True)
```

```
sld_file.close()
```

create_workspace (*workspace_id*, *uri*, *debug=False*)

Create a new workspace.

Parameters

- **workspace_id** (*string*) – Identifier of the workspace to create. Must be unique.
- **uri** (*string*) – URI associated with your project. Does not need to be a real web URL, just a unique identifier. One suggestion is to append the URL of your project with the name of the workspace (e.g.: <http://www.example.com/workspace-name>).
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.create_workspace(workspace_id='workspace_name',
uri='www.example.com/workspace_name')
```

delete_layer (*layer_id*, *purge=False*, *recurse=False*, *debug=False*)

Delete a layer.

Parameters

- **layer_id** (*string*) – Identifier of the layer to delete.
- **purge** (*bool, optional*) – Purge if True.
- **recurse** (*bool, optional*) – Delete recursively if True (i.e: delete layer groups it belongs to).
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.delete_layer('workspace:layer_name')
```

delete_layer_group (*layer_group_id*, *purge=False*, *recurse=False*, *debug=False*)

Delete a layer group.

Parameters

- **layer_group_id** (*string*) – Identifier of the layer group to delete.
- **purge** (*bool, optional*) – Purge if True.
- **recurse** (*bool, optional*) – Delete recursively if True.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.delete_layer_group('layer_group_name')
```

delete_resource (*resource_id*, *store=None*, *purge=False*, *recurse=False*, *debug=False*)

Delete a resource.

Parameters

- **resource_id** (*string*) – Identifier of the resource to delete.
- **store** (*string*, *optional*) – Delete resource from this store.
- **purge** (*bool*, *optional*) – Purge if True.
- **recurse** (*bool*, *optional*) – Delete recursively any dependencies if True (i.e.: layers or layer groups it belongs to).
- **debug** (*bool*, *optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.delete_resource('workspace:resource_name')
```

delete_store (*store_id*, *purge=False*, *recurse=False*, *debug=False*)

Delete a store.

Parameters

- **store_id** (*string*) – Identifier of the store to delete.
- **purge** (*bool*, *optional*) – Purge if True.
- **recurse** (*bool*, *optional*) – Delete recursively if True.
- **debug** (*bool*, *optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.delete_store('workspace:store_name')
```

delete_style (*style_id*, *purge=False*, *recurse=False*, *debug=False*)

Delete a style.

Parameters

- **style_id** (*string*) – Identifier of the style to delete.
- **purge** (*bool*, *optional*) – Purge if True.

- **recurse** (*bool, optional*) – Delete recursively if True.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.delete_resource('style_name')
```

delete_workspace (*workspace_id, purge=False, recurse=False, debug=False*)

Delete a workspace.

Parameters

- **workspace_id** (*string*) – Identifier of the workspace to delete.
- **purge** (*bool, optional*) – Purge if True.
- **recurse** (*bool, optional*) – Delete recursively if True.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.delete_resource('workspace_name')
```

get_layer (*layer_id, debug=False*)

Retrieve a layer object.

Parameters

- **layer_id** (*string*) – Identifier of the layer to retrieve. Can be a name or a workspace-name combination (e.g.: “name” or “workspace:name”).
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.get_layer('layer_name')
```

```
response = engine.get_layer('workspace_name:layer_name')
```

get_layer_group (*layer_group_id, debug=False*)

Retrieve a layer group object.

Parameters

- **layer_group_id** (*string*) – Identifier of the layer group to retrieve. Can be a name or a workspace-name combination (e.g.: “name” or “workspace:name”).
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.get_layer_group('layer_group_name')
```

```
response = engine.get_layer_group('workspace_name:layer_group_name')
```

get_resource (*resource_id, store=None, debug=False*)

Retrieve a resource object.

Parameters

- **resource_id** (*string*) – Identifier of the resource to retrieve. Can be a name or a workspace-name combination (e.g.: “name” or “workspace:name”).
- **store** (*string, optional*) – Get resource from this store.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.get_resource('example_workspace:resource_name')
```

```
response = engine.get_resource('resource_name', store='example_store')
```

get_store (*store_id, debug=False*)

Retrieve a store object.

Parameters

- **store_id** (*string*) – Identifier of the store to retrieve. Can be a name or a workspace-name combination (e.g.: “name” or “workspace:name”).
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.get_store('store_name')
```

```
response = engine.get_store('workspace_name:store_name')
```

get_style (*style_id*, *debug=False*)

Retrieve a style object.

Parameters

- **style_id** (*string*) – Identifier of the style to retrieve.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.get_style('style_name')
```

get_workspace (*workspace_id*, *debug=False*)

Retrieve a workspace object.

Parameters

- **workspace_id** (*string*) – Identifier of the workspace to retrieve.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.get_workspace('workspace_name')
```

list_layer_groups (*with_properties=False*, *debug=False*)

List the names of all layer groups available from the spatial dataset service.

Parameters

- **with_properties** (*bool, optional*) – Return list of layer group dictionaries instead of a list of layer group names.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.list_layer_groups()
```

```
response = engine.list_layer_groups(with_properties=True)
```

list_layers (*with_properties=False*, *debug=False*)

List names of all layers available from the spatial dataset service.

Parameters

- **with_properties** (*bool, optional*) – Return list of layer dictionaries instead of a list of layer names.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.list_layers()
```

```
response = engine.list_layers(with_properties=True)
```

list_resources (*with_properties=False, store=None, workspace=None, debug=False*)

List the names of all resources available from the spatial dataset service.

Parameters

- **with_properties** (*bool, optional*) – Return list of resource dictionaries instead of a list of resource names.
- **store** (*string, optional*) – Return only resources belonging to a certain store.
- **workspace** (*string, optional*) – Return only resources belonging to a certain workspace.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.list_resource()
```

```
response = engine.list_resource(store="example_store")
```

```
response = engine.list_resource(with_properties=True, workspace="example_workspace")
```

list_stores (*workspace=None, with_properties=False, debug=False*)

List the names of all stores available from the spatial dataset service.

Parameters

- **workspace** (*string, optional*) – List long stores belonging to this workspace.
- **with_properties** (*bool, optional*) – Return list of store dictionaries instead of a list of store names.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.list_stores()
```

```
response = engine.list_stores(workspace='example_workspace', with_properties=True)
```

list_styles (*with_properties=False, debug=False*)

List the names of all styles available from the spatial dataset service.

Parameters

- **with_properties** (*bool, optional*) – Return list of style dictionaries instead of a list of style names.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.list_styles()
```

```
response = engine.list_styles(with_properties=True)
```

list_workspaces (*with_properties=False, debug=False*)

List the names of all workspaces available from the spatial dataset service.

Parameters

- **with_properties** (*bool, optional*) – Return list of workspace dictionaries instead of a list of workspace names.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.list_workspaces()
```

```
response = engine.list_workspaces(with_properties=True)
```

type

GeoServer Spatial Dataset Type

update_layer (*layer_id, debug=False, **kwargs*)

Update an existing layer.

Parameters

- **layer_id** (*string*) – Identifier of the layer to update.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.

- ****kwargs** (*kwargs, optional*) – Key value pairs representing the attributes and values to change.

Returns Response dictionary

Return type (dict)

Examples

```
updated_layer = engine.update_layer(layer_id='workspace:layer_name', default_style='style1',
styles=['style1', 'style2'])
```

update_layer_group (*layer_group_id, debug=False, **kwargs*)

Update an existing layer. If modifying the layers, ensure the number of layers and the number of styles are the same.

Parameters

- **layer_group_id** (*string*) – Identifier of the layer group to update.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.
- ****kwargs** (*kwargs, optional*) – Key value pairs representing the attributes and values to change

Returns Response dictionary

Return type (dict)

Examples

```
updated_layer_group = engine.update_layer_group(layer_group_id='layer_group_name', layers=['layer1', 'layer2'], styles=['style1', 'style2'])
```

update_resource (*resource_id, store=None, debug=False, **kwargs*)

Update an existing resource.

Parameters

- **resource_id** (*string*) – Identifier of the resource to update. Can be a name or a workspace-name combination (e.g.: “name” or “workspace:name”).
- **store** (*string, optional*) – Update a resource in this store.
- **debug** (*bool, optional*) – Pretty print the response dictionary to the console for debugging. Defaults to False.
- ****kwargs** (*kwargs, optional*) – Key value pairs representing the attributes and values to change.

Returns Response dictionary

Return type (dict)

Examples

```
response = engine.update_resource(resource_id='workspace:resource_name', enabled=False, title='New Title')
```

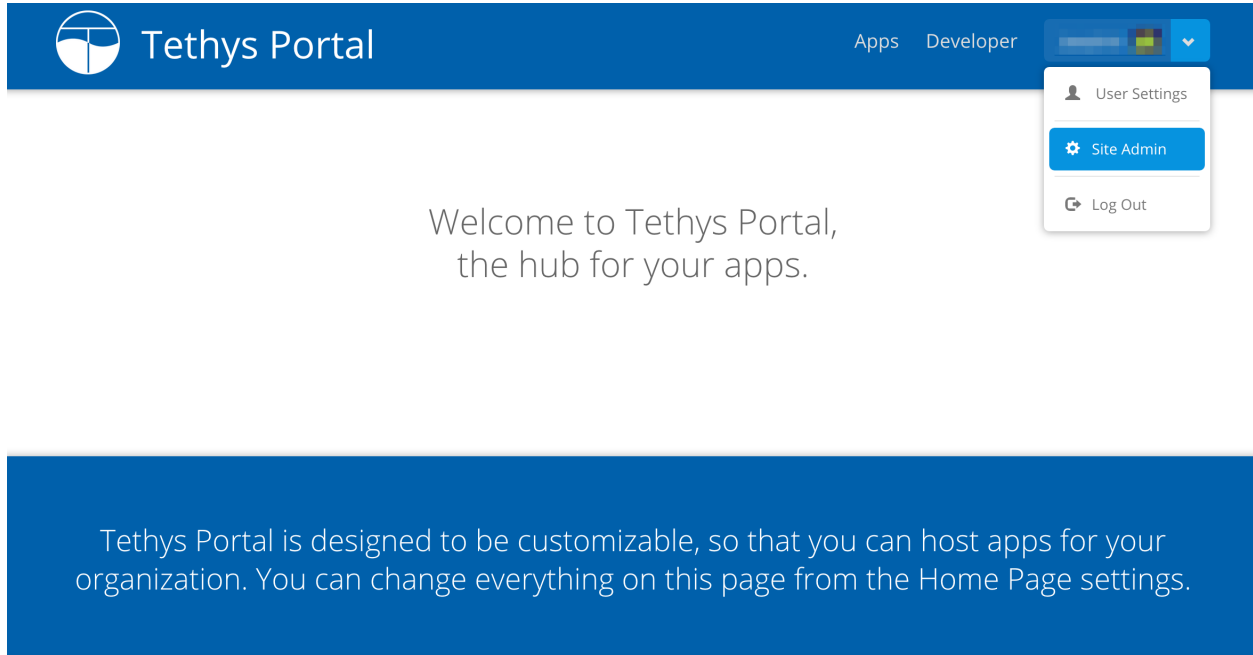
`validate()`

Validate the GeoServer spatial dataset engine. Will throw an error if not valid.

Register New Spatial Dataset Services

Registering new spatial dataset services is performed using the System Admin Settings.

1. Login to your Tethys Platform instance as an administrator.
2. Select “Site Admin” from the user drop down menu.



3. Select “Spatial Dataset Services” from the “Tethys Services” section.
4. Select an existing Spatial Dataset Service configuration from the list to edit it OR click on the “Add Spatial Dataset Service” button to create a new one.
5. Give the Spatial Dataset Service configuration a name, select an appropriate engine, specify the endpoint, and provide a username and password. The name of the configuration must be unique, because it is used to retrieve the Spatial Dataset Service connection object. The endpoint is a URL pointing to the GeoServer REST endpoint. This endpoint can be for **any** GeoServer. If you want to use the built-in GeoServer installation, you can obtain the endpoint by running `tethys docker ip` in a terminal:

```
$ tethys docker ip
...
```

Tethys Portal Apps Developer

Site Administration

Authentication and Authorization	
Groups	+ Add Change
Users	+ Add Change

Tethys Compute	
Clusters	+ Add Change
Settings	Change

Tethys Portal	
Site Settings	Change

Tethys Services	
Dataset Services	+ Add Change
Spatial Dataset Services	+ Add Change
Web Processing Services	+ Add Change

Recent Actions
My Actions
[General Settings](#)
Settings Category
[default](#)
Web Processing Service
[default_geoserver](#)
Spatial Dataset Service
[default](#)
Web Processing Service
[default_geoserver](#)
Spatial Dataset Service
[default_geoserver](#)
Spatial Dataset Service
[default](#)
Dataset Service
[default](#)
Dataset Service
[default_ckan](#)
Dataset Service
[default](#)
Dataset Service

Copyright © 2015 Your Organization Powered by Tethys Platform

Tethys Portal Apps Developer

[Home](#) > [Tethys Services](#) > [Spatial Dataset Services](#)

Select Spatial Dataset Service To Change

[Add Spatial Dataset Service +](#)

Action: [Go](#) 0 of 1 selected

<input type="checkbox"/> Spatial Dataset Service
<input type="checkbox"/> default_geoserver

1 Spatial Dataset Service

Copyright © 2015 Your Organization Powered by Tethys Platform

```

GeoServer:
  Host: localhost
  Port: 8181
  Endpoint: http://localhost:8181/geoserver/rest

```

...

When you are done, your Spatial Dataset Service configuration should look similar to this:

The screenshot shows the Tethys Portal interface for configuring a Spatial Dataset Service. The page title is "Change Spatial Dataset Service". The configuration form includes the following fields:

- Name:** default_geoserver
- Engine:** GeoServer
- Endpoint:** http://192.168.59.103:8181/geoserver/res
- Apikey:** (empty)
- Username:** admin
- Password:** (masked with dots)

At the bottom of the form, there are four buttons: "Delete" (with a red asterisk icon), "Save and add another", "Save and continue editing", and "Save" (in a green box). A "History" button is also visible in the top right corner of the form area.

6. Press “Save” to save the Dataset Service configuration.

Note: Prior to version Tethys Platform 1.1.0, it was possible to register spatial dataset services using a mechanism in the *app configuration file*. This mechanism has been deprecated due to security concerns.

Working with Spatial Dataset Services

After spatial dataset services have been properly configured, you can use the services to store, publish, and retrieve data for your apps. This process typically involves the following steps:

1. Get a Spatial Dataset Engine

The Spatial Dataset Services API provides a convenience function called `get_spatial_dataset_engine`. To retrieve an engine for a sitewide configuration, call `get_spatial_dataset_engine` with the name of the configuration:

```
from tethys_apps.sdk import get_spatial_dataset_engine
```

```
dataset_engine = get_dataset_engine(name='example')
```

It will return the first service with a matching name or raise an exception if the service cannot be found with the given name. Alternatively, you may retrieve a list of all the spatial dataset engine objects that are registered using the `list_spatial_dataset_engines` function:

```
from tethys_apps.sdk import list_spatial_dataset_engines
```

```
dataset_engines = list_spatial_dataset_engines()
```

You can also create a `SpatialDatasetEngine` object directly without using the convenience function. This can be useful if you want to vary the credentials for dataset access frequently (e.g.: using user specific credentials). Simply import it and instantiate it with valid credentials:

```
from tethys_dataset_services.engines import GeoServerSpatialDatasetEngine
```

```
spatial_dataset_engine = GeoServerSpatialDatasetEngine(
```

```
endpoint='http://www.example.com/api/3/action'
```

Caution: Take care not to store API keys, usernames, or passwords in the source files of your app—especially if the source is made public. This could compromise the security of the spatial dataset service.

2. Use the Spatial Dataset Engine

After you have a `SpatialDatasetEngine` object, simply call the desired method on it. All `SpatialDatasetEngine` methods return a dictionary with an item named ‘success’ that contains a boolean. If the operation was successful, ‘success’ will be true, otherwise it will be false. If ‘success’ is true, the dictionary will have an item named ‘result’ that will contain the results. If it is false, the dictionary will have an item named ‘error’ that will contain information about the error that occurred. This can be very useful for debugging and error catching purposes.

Consider the following example for uploading a shapefile to spatial dataset services:

```
from tethys_apps.sdk import get_spatial_dataset_engine
```

```
# First get an engine
```

```
engine = get_spatial_dataset_engine(name='example')
```

```
# Create a workspace named after our app
```

```
engine.create_workspace(workspace_id='my_app', uri='http://www.example.com/apps/my-app')
```

```
# Path to shapefile base for foo.shp, side cars files (e.g.: .shx, .dbf) will be  
# gathered in addition to the .shp file.
```

```
shapefile_base = '/path/to/foo'
```

```
# Notice the workspace in the store_id parameter
```

```
result = dataset_engine.create_shapefile_resource(store_id='my_app:foo',
```

```
shapefile_base=shapefile_base)
```

```
# Chraise if it was successful
```

```
if not result['success']:
```

A new shapefile Data Store will be created called ‘foo’ in workspace ‘my_app’ and a resource will be created for the shapefile called ‘foo’. A layer will also automatically be configured for the new shapefile resource.

Tip: When you are learning how to use the spatial dataset engine methods, run the commands with the debug parameter set to true. This will automatically pretty print the result dictionary to the console so that you can inspect its contents:

```
# Example method with debug option
engine.list_layers(debug=True)
```

3. Get OGC Web Service URL

Publishing the spatial dataset with a spatial dataset service would be pointless without using the service to render the data on a map. This can be done by querying the data using the OGC web services WFS, WCS, or WMS. The dictionary that is returned when retrieving layers, layer groups, or resources will include a key for appropriate OGC services for the object returned. Feature type resources will provide a “wfs” key, coverage resources will provide a “wcs” key, and layers and layergroups will provide a “wms” key. The value will be another dictionary of OGC queries for different endpoints. For example:

```
# Get a feature type layer
response = engine.get_layer(layer_id='sf:roads', debug=True)

# Response dictionary includes "wms" key with links to maps in various formats
{'result': {'advertised': True,
            'attribution': None,
            'catalog': 'http://localhost:8181/geoserver/',
            'default_style': 'simple_roads',
            'enabled': None,
            'href': 'http://localhost:8181/geoserver/rest/layers/sf%3Aroads.xml',
            'name': 'sf:roads',
            'resource': 'sf:roads',
            'resource_type': 'layer',
            'styles': ['sf:line'],
            'wms': {...}
            'success': True}
```

These links could be passed on to a web mapping client like OpenLayers or Google Maps to render the map interactively on a web page. Note that the OGC mapping services are very powerful and the links provided represent only a simple query. You can construct custom OGC URLs queries without much difficulty. For excellent primers on WFS, WCS, and WMS with GeoServer, visit these links:

- [GeoServer Web Feature Service Overview](#)
- [GeoServer Web Coverage Service Overview](#)
- [GeoServer Web Map Service Overview](#)

1.6.9 Web Processing Services API

Last Updated: May 13, 2015

Web Processing Services (WPS) are web services that can be used perform geoprocessing and other processing activities for apps. The Open Geospatial Consortium (OGC) has created the [WPS interface standard](#) that provides rules for how inputs and outputs for processing services should be handled. Using the Web Processing Services API, you will be able to provide processing capabilities for your apps using any service that conforms to the OGC WPS standard. For convenience, the 52 North WPS is provided as part of the Tethys Platform software suite. Refer to the [Installation](#) documentation to learn how to install Tethys Platform with 52 North WPS enabled.

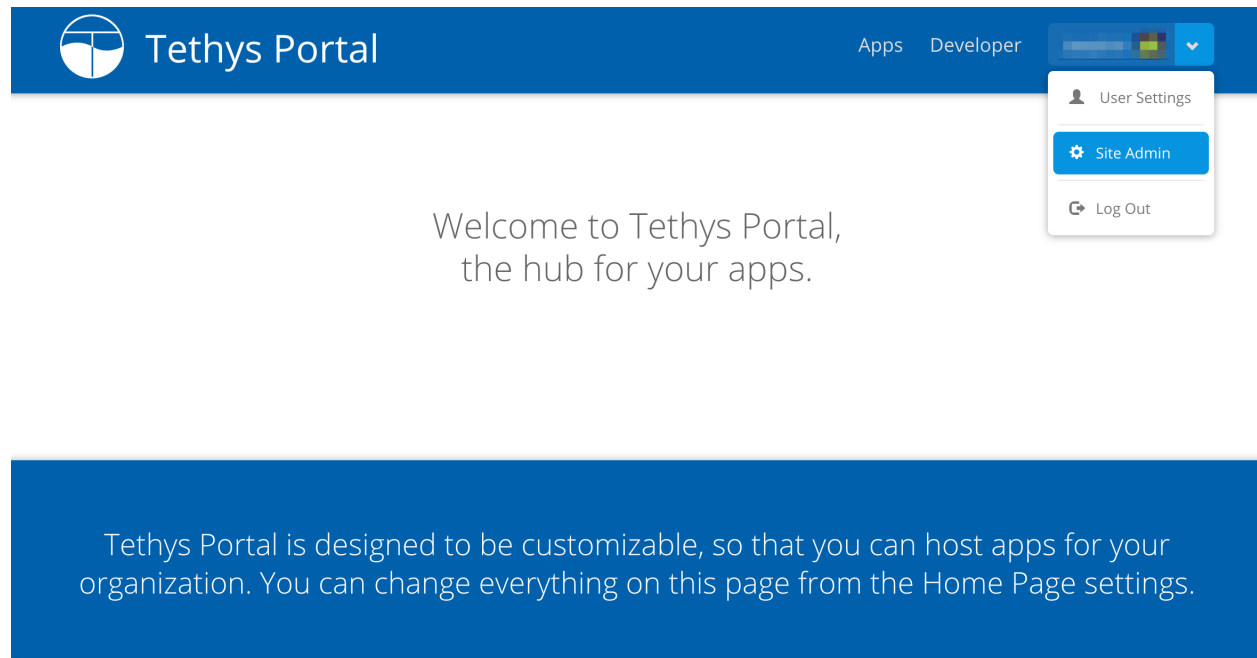
Configuring WPS Services

Before you can start using WPS services in your apps, you will need link your Tethys Platform to a valid WPS. This can be done either at a sitewide level or at an app specific level. When a WPS is configured at the sitewide level, all apps that are installed on that Tethys Platform instance will be able to access the WPS. When installed at an app specific level, the WPS will only be accessible to the app that it is linked to. The following sections will describe how to configure a WPS to be used at both of these levels.

Register New WPS Service

Sitewide configuration is performed using the System Admin Settings.

1. Login to your Tethys Platform instance as an administrator.
2. Select “Site Admin” from the user drop down menu.



localhost:8000/admin/

3. Select “Web Processing Services” from the “Tethys Services” section.

The screenshot displays the Tethys Portal Site Administration interface. The main content area is divided into several sections:

- Authentication and Authorization:** Includes 'Groups' and 'Users', each with 'Add' and 'Change' buttons.
- Tethys Compute:** Includes 'Clusters' and 'Settings', each with 'Add' and 'Change' buttons.
- Tethys Portal:** Includes 'Site Settings' with a 'Change' button.
- Tethys Services:** Includes 'Dataset Services', 'Spatial Dataset Services', and 'Web Processing Services', each with 'Add' and 'Change' buttons.

The 'Recent Actions' sidebar on the right shows a list of actions performed, including 'General Settings', 'default', 'default_geoserver', and 'default_ckan'.

At the bottom of the page, the footer contains the text: "Copyright © 2015 Your Organization" and "Powered by Tethys Platform".

4. Select an existing Web Processing Service configuration from the list to edit it OR click on the “Add Web Processing Service” button to create a new one.
5. Give the Web Processing Service configuration a name and specify the endpoint. The name must be unique, because it is used to connect to the WPS. The endpoint is a URL pointing to the WPS. For example, the endpoint for the 52 North WPS demo server would be:

```
http://geoprocessing.demo.52north.org:8080/wps/WebProcessingService
```



If authentication is required, specify the username and password.

6. Press “Save” to save the WPS configuration.

Note: Prior to version Tethys Platform 1.1.0, it was possible to register WPS services using a mechanism in the *app configuration file*. This mechanism has been deprecated due to security concerns.

Working with WPS Services in Apps

The Web Processing Service API is powered by [OWSLib](#), a Python client that can be used to interact with OGC web services. For detailed explanations the WPS client provided by OWSLib, refer to the [OWSLib WPS Documentation](#). This article only provides a basic introduction to working with the OWSLib WPS client.

 Tethys Portal Apps Developer 

[Home](#) > [Tethys Services](#) > [Web Processing Services](#)


Select Web Processing Service To Change



[Add Web Processing Service +](#)

Action: 0 of 2 selected

<input type="checkbox"/>	Web Processing Service
<input type="checkbox"/>	default_wps
<input type="checkbox"/>	default

2 Web Processing Services

Copyright © 2015 Your Organization Powered by  Tethys Platform

 Tethys Portal Apps Developer 

[Home](#) > [Tethys Services](#) > [Web Processing Services](#) > default_wps

Change Web Processing Service

[History](#)


Name:

Endpoint:

Username:

Password:

[✖ Delete](#)

Copyright © 2015 Your Organization Powered by  Tethys Platform

Get a WPS Engine

Anytime you wish to use a WPS service in an app, you will need to obtain an `owslib.wps.WebProcessingService` engine object. The Web Processing Service API provides a convenience function for retrieving `owslib.wps.WebProcessingService` engine objects called `get_wps_service_engine`. Basic usage involves calling the function with the name of the WPS service that you wish to use. For example:

```
from tethys_apps.sdk import get_wps_service_engine

wps_engine = get_wps_service_engine(name='example')
```

Alternatively, you may retrieve a list of all the dataset engine objects that are registered using the `list_wps_service_engines` function:

```
from tethys_apps.sdk import list_wps_service_engines

wps_engines = list_wps_service_engines()
```

You can also create an `owslib.wps.WebProcessingService` engine object directly without using the convenience function. This can be useful if you want to vary the credentials for WPS service access frequently (e.g.: using user specific credentials).

```
from owslib.wps import WebProcessingService

wps_engine = WebProcessingService('http://www.example.com/wps/WebProcessingService',
verbose=False, skip_caps=wps_engine.getcapabilities())
```

Using the WPS Engine

After you have retrieved a valid `owslib.wps.WebProcessingService` engine object, you can use it execute process requests. The following example illustrates how to execute the GRASS buffer process on a 52 North WPS:

```
from owslib.wps import GMLMultiPolygonFeatureCollection

polygon = [(-102.8184, 39.5273), (-102.8184, 37.418),
(-101.2363, 37.418), (-101.2363, 39.5273),
(-102.8184, 39.5273)]
feature_collection = GMLMultiPolygonFeatureCollection( [polygon] )
process_id = 'v.buffer'
inputs = [ ('DISTANCE', 5.0),
('INPUT', feature_collection)
]
output = 'OUTPUT'
execution = wps_engine.execute(process_id, inputs, output)
monitorExecution(execution)
```

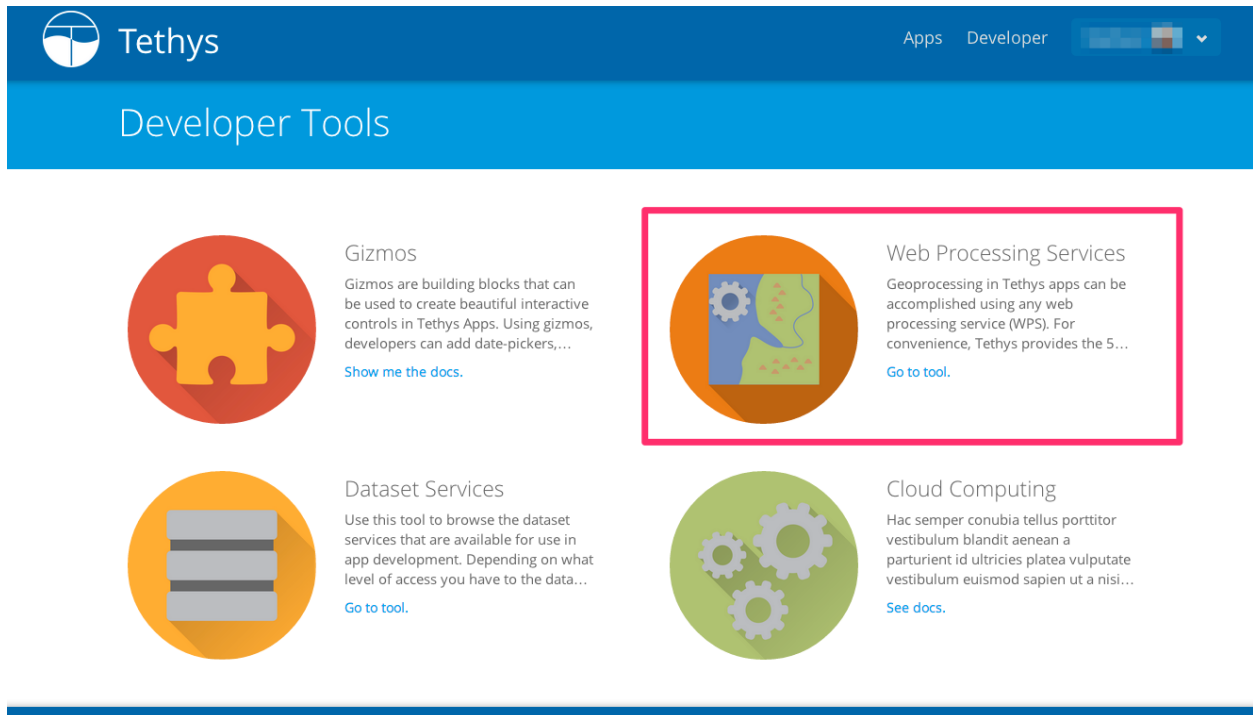
It is also possible to perform requests using data that are hosted on WFS servers, such as the GeoServer that is provided as part of the Tethys Platform software suite. See the [OWSLib WPS Documentation](#) for more details on how this is to be done.

Web Processing Service Developer Tool

Tethys Platform provides a developer tool that can be used to browse the sitewide WPS services and the processes that they provide. This tool is useful for formulating new process requests. To use the tool:

1. Browse to the Developer Tools page of your Tethys Platform by selecting the “Developer” link from the menu at the top of the page.

2. Select the tool titled “Web Processing Services”.



3. Select a WPS service from the list of services that are linked with your Tethys Instance. If no WPS services are linked to your Tethys instance, follow the steps in Site-wide Configuration, above, to setup a WPS service.
4. Select the process you wish to view.

A description of the process and the inputs and outputs will be displayed.

1.6.10 Distributed Computing API

Last Updated: May 26, 2015

Distributed computing in Tethys Platform is made possible with HTCondor. HTCondor computing resources are managed through the Tethys Compute settings of the site admin in Tethys Portal. Access to the HTCondor computing environment is made possible through a JobManager object.

Tethys Compute

The Tethys Compute heading in site admin allows an administrator to configure settings for computing resources and to manage computing clusters.



Web Processing Services

This tool can be used to browse the available processing capabilities of any Web Processing Services (WPS) that are linked with this instance of Tethys Platform.

Linked WPS Services

52°North WPS 3.3.1

Service based on the 52°North implementation of WPS 1.0.0

Provider: 52North

Type: WPS

Version: 1.0.0



52°North WPS 3.3.1

Service based on the 52°North implementation of WPS 1.0.0

Processes

buffer

org.n52.wps.server.algorithm.SimpleBufferAlgorithm

org.n52.wps.server.algorithm.SimpleBufferAlgorithm

v.buffer

Creates a buffer around vector features of given type.



v.buffer

Creates a buffer around vector features of given type.

<http://grass.osgeo.org/grass70/manuals/v.buffer.html>

Input

input (ComplexData): Name of input vector map

REQUIRED

Or data source for direct OGR access

Min. Occurrences: 1

Max. Occurrences: 1

Default Value:

Complex Data

Schema: <http://schemas.opengis.net/gml/3.1.1/base/gml.xsd>

MIME Type: text/xml

Encoding: UTF-8

Supported Values:

Complex Data

Schema: <http://schemas.opengis.net/gml/3.1.1/base/gml.xsd>

MIME Type: text/xml

Encoding: UTF-8

Complex Data

Schema: <http://schemas.opengis.net/gml/3.1.1/base/gml.xsd>

MIME Type: application/xml

Encoding: UTF-8

Complex Data

Schema: <http://schemas.opengis.net/gml/2.1.2/feature.xsd>

MIME Type: text/xml

Encoding: UTF-8

Settings

Cluster Management

Scheduler IP

Scheduler Key Location

Default Cluster

Amazon Credentials

AWS Access Key ID

AWS Secret Access Key

AWS User ID

Key Name

Key Location

Azure Credentials

Subscription ID

Certificate Path

Clusters

- uses TethysCluster
- able to provision clusters on AWS and Azure
- Describe settings

Job Manager

Configuring the Job Manager

Using the Job Manager in your App

1.6.11 Command Line Interface

Last Updated: November 18, 2014

The Tethys Command Line Interface (CLI) provides several commands that are used for managing Tethys Platform and Tethys apps. The *Python virtual environment* must be activated to use the command line tools. This can be done using the following command:

```
$ ./usr/lib/tethys/bin/activate
```

The following article provides an explanation for each command provided by Tethys CLI.

Usage

```
$ tethys <command> [options]
```

Options

- **-h, --help**: Request the help information for Tethys CLI or any command.

Commands

scaffold <name>

This command is used to create new Tethys app projects via the scaffold provided by Tethys Platform. You will be presented with several interactive prompts requesting metadata information that can be included with the app. The new app project will be created in the current working directory of your terminal.

Arguments:

- **name**: The name of the new Tethys app project to create. Only lowercase letters, numbers, and underscores are allowed.

Examples:

```
$ tethys scaffold my_first_app
```

gen <type>

Aids the installation of Tethys by automating the creation of supporting files.

Arguments:

- **type**: The type of object to generate. Either “settings” or “apache”.
 - *settings*: When this type of object is specified, **gen** will generate a new `settings.py` file. It generates the `settings.py` with a new `SECRET_KEY` each time it is run.
 - *apache*: When this type of object is specified **gen** will generate a new `apache.conf` file. This file is used to configure Tethys Platform in a production environment.

Optional Arguments:

- **-d DIRECTORY, --directory DIRECTORY**: Destination directory for the generated object.

Examples:

```
$ tethys gen settings
$ tethys gen settings -d /path/to/destination
$ tethys gen apache
$ tethys gen apache -d /path/to/destination
```

manage <subcommand> [options]

This command contains several subcommands that are used to help manage Tethys Platform.

Arguments:

- **subcommand**: The management command to run. Either “start”, “syncdb”, or “collectstatic”.
 - *start*: Starts the Django development server. Wrapper for `manage.py runserver`.
 - *syncdb*: Initialize the database during installation. Wrapper for `manage.py syncdb`.
 - *collectstatic*: Link app static/public directories to `STATIC_ROOT` directory and then run Django’s `collectstatic` command. Preprocessor and wrapper for `manage.py collectstatic`.

Optional Arguments:

- **-p PORT, --port PORT**: Port on which to start the development server. Default port is 8000.
- **-m MANAGE, --manage MANAGE**: Absolute path to `manage.py` file for Tethys Platform installation if different than default.

Examples:

```
# Start the development server
$ tethys manage start
$ tethys manage start -p 8888

# Sync the database
$ tethys manage syncdb

# Collect static files
$ tethys manage collectstatic
```

syncstores <app_name, app_name...> [options]

Management command for Persistent Stores. To learn more about persistent stores see [Persistent Stores API](#).

Arguments:

- **app_name**: Name of one or more apps to target when performing persistent store sync OR “all” to sync all persistent stores on this Tethys Platform instance.

Optional Arguments:

- **-r, --refresh**: Drop databases prior to performing persistent store sync resulting in a refreshed database.
- **-f, --firsttime**: All initialization functions will be executed with the `first_time` parameter set to `True`.
- **-d, DATABASE, --database DATABASE**: Name of the persistent store database to target.
- **-m MANAGE, --manage MANAGE**: Absolute path to `manage.py` file for Tethys Platform installation if different than default.

Examples:

```
# Sync all persistent store databases for one app
$ tethys syncstores my_first_app

# Sync all persistent store databases for multiple apps
$ tethys syncstores my_first_app my_second_app yet_another_app

# Sync all persistent store databases for all apps
$ tethys syncstores all

# Sync a specific persistent store database for an app
$ tethys syncstores my_first_app -d example_db

# Sync persistent store databases with a specific name for all apps
$ tethys syncstores all -d example_db

# Sync all persistent store databases for an app and force first_time to True
$ tethys syncstores my_first_app -f

# Refresh all persistent store databases for an app
$ tethys syncstores my_first_app -r
```

uninstall <app>

Use this command to uninstall apps.

Arguments:

- **app**: Name the app to uninstall.

Examples:

```
# Uninstall my_first_app
$ tethys uninstall my_first_app
```

docker <subcommand> [options]

Management commands for the Tethys Docker containers. To learn more about Docker, see [What is Docker?](#).

Arguments:

- **subcommand**: The docker command to run. One of the following:
 - *init*: Initialize the Tethys Dockers including, starting Boot2Docker if applicable, pulling the Docker images, and installing/creating the Docker containers.
 - *start*: Start the Docker containers.
 - *stop*: Stop the Docker containers.
 - *restart*: Restart the Docker containers.
 - *status*: Display status of each Docker container.
 - *update*: Pull the latest version of the Docker images.
 - *remove*: Remove a Docker images.
 - *ip*: Display host, port, and endpoint of each Docker container.

Optional Arguments:

- **-d, --defaults:** Install Docker containers with default values (will not prompt for input). Only applicable to *init* subcommand.
- **-c {postgis, geoserver, wps}, --container {postgis, geoserver, wps}:** Execute subcommand only on the container specified.
- **-b, --boot2docker:** Also stop Boot2Docker when *stop* subcommand is called with this option.

Examples:

```
# Initialize Tethys Dockers
$ tethys docker init

# Initialize with Default Parameters
$ tethys docker init -d

# Start all Tethys Dockers
$ tethys docker start

# Start only PostGIS Docker
$ tethys docker start -c postgis

# Stop Tethys Dockers
$ tethys docker stop

# Stop Tethys Dockers and Boot2Docker if applicable
$ tethys docker stop -b

# Update Tethys Docker Images
$ tethys docker update

# Remove Tethys Docker Images
$ tethys docker remove

# View Status of Tethys Dockers
$ tethys docker status

# View Host and Port Info
$ tethys docker ip
```

1.7 Tethys Portal

Last Updated: May 23, 2015

Tethys Portal is the Django web site provided by Tethys Platform that acts as the runtime environment for apps. It leverages the capabilities of Django to provide the core website functionality that is often taken for granted in modern web applications. A description of the primary capabilities of Tethys Portal is provided in this article.

1.7.1 Administrator Pages

Tethys Portal includes administration pages that can be used to manage the website (see Figure 1). The administration dashboard is only available to administrator users. You should have created a default administrator user when you installed Tethys Platform. If you are logged in as an administrator, you will be able to access the administrator dashboard by selecting the “Site Admin” option from the user drop down menu in the top right-hand corner of the page.

Figure 1. Administrator dashboard for Tethys Portal.

Note: If you did not create an administrator user during installation, run the following command in the terminal:

```
$ python /usr/lib/tethys/src/manage.py createsuperuser
```

1.7.2 Manage Users and Permissions

Permissions and users can be managed from the administrator dashboard using the `Groups` and `Users` links under the `Authentication and Authorization` heading. Figure 4 shows an example of the user management page for a user named John. Permissions can be assigned to each user individually or users can be assigned to groups and they will be given the permissions of that group.

Figure 4. User management for Tethys Portal.

1.7.3 Customize

The content of Tethys Portal can be customized or rebranded to reflect your organization. To access these settings, login to Tethys Portal using an administrator account and select the `Site Settings` link under the `Tethys Portal` heading. Sitewide settings can be changed using the `General Settings` link (Figure 2) and the content on the home page can be modified by using the `Home Page` link. The general settings that are available include the following:

- **Site Title:** the title of the web page that appears in browser tabs and bookmarks of the site.
- **Favicon:** the path to the image that is used in browser tabs and bookmarks.
- **Brand Text:** the title that appears in the header.

 Tethys Apps Developer John  ▾

[Home](#) > [Authentication and Authorization](#) > [Users](#) > john

Change User

[History](#)

Username:
Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.

Password: **algorithm:** pbkdf2_sha256 **iterations:** 12000 **salt:** 5kjUht***** **hash:** SvXTQb*****
Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using [this form](#).

Personal Info

First name:

Last name:

Email address:


Permissions


Active
Designates whether this user should be treated as active. Unselect this instead of deleting accounts.


Staff status
Designates whether the user can log into this admin site.

Superuser status
Designates that this user has all permissions without explicitly assigning them.

The groups this user belongs to. A user will get all permissions granted to each of his/her group. Hold down "Control", or "Command" on a Mac, to select more than one.

Groups: 

Available groups 

Chosen groups 

- Brand Image: the logo or image that appears next to the title in the header.
- Apps Library Title: the title of the page that displays app icons.
- Primary Color: color that is used as the primary theme color (e.g.: #ff0000 or rgb(255,0,0)).
- Secondary Color: color that is used as the secondary theme color (e.g.: #ff0000 or rgb(255,0,0)).
- Footer Copyright: The copyright text that appears in the footer.

The screenshot shows the Tethys Portal interface. At the top, there is a blue header with the Tethys logo and navigation links for 'Apps' and 'Developer'. Below the header, a breadcrumb trail reads 'Home > Tethys Portal > Site Settings > General Settings'. The main heading is 'Change Settings Category' with a 'History' button to its right. Underneath, the 'General Settings' section contains a table with the following data:

Name	Content	Date modified
Site Title	Tethys	Feb. 6, 2015, 2:50 a.m.
Favicon	/static/tethys_portal/images/default_favicon.png	Feb. 6, 2015, 2:50 a.m.
Brand Text	Tethys	Feb. 6, 2015, 2:50 a.m.
Brand Image	/static/tethys_portal/images/tethys-logo-75.png	Feb. 6, 2015, 2:50 a.m.
Primary Color	#0a62a9	Feb. 6, 2015, 4:52 a.m.
Secondary Color	#1B95DC	Feb. 6, 2015, 4:52 a.m.

At the bottom right of the table, there are two buttons: 'Save and continue editing' and a green 'Save' button.

Figure 2. General settings for Tethys Portal.

Figure 3 shows a screenshot of the home page settings that are available. The settings that can be modified on the home page include:

- Hero Text
- Blurb Text
- Feature 1 Heading
- Feature 1 Body
- Feature 1 Image
- Feature 2 Heading
- Feature 2 Body
- Feature 2 Image
- Feature 3 Heading

- Feature 3 Body
- Feature 3 Image
- Call to Action
- Call to Action Button

Feature 1 Image	<input type="text" value="/static/tethys_portal/images/placeholder.gif"/>	Feb. 6, 2015, 2:50 a.m.
Feature 2 Heading	<input type="text" value="Feature 2"/>	Feb. 6, 2015, 2:50 a.m.
Feature 2 Body	<input type="text" value="Describe the apps and tools that your Tethys Portal provides and add custom pictures to each feature as a finishing touch."/>	Feb. 6, 2015, 2:50 a.m.
Feature 2 Image	<input type="text" value="/static/tethys_portal/images/placeholder.gif"/>	Feb. 6, 2015, 2:50 a.m.
Feature 3 Heading	<input type="text" value="Feature 3"/>	Feb. 6, 2015, 2:50 a.m.
Feature 3 Body	<input type="text" value="You can change the color theme and branding of your Tethys Portal in a jiffy. Visit the Site Admin settings from the user menu and select General Settings."/>	Feb. 6, 2015, 2:50 a.m.
Feature 3 Image	<input type="text" value="/static/tethys_portal/images/placeholder.gif"/>	Feb. 6, 2015, 2:50 a.m.
Call to Action	<input type="text" value="Ready to get started?"/>	Feb. 6, 2015, 2:50 a.m.
Call to Action Button	<input type="text" value="Start Using Tethys!"/>	Feb. 6, 2015, 2:50 a.m.

Figure 3. Home page settings for Tethys Portal.

Tethys Portal can also be configured to bypass the home page. When this setting is applied, the root url will always redirect to the apps library page. This setting is modified in the `settings.py` script. Simply set the `BYPASS_TETHYS_HOME_PAGE` setting to `True` like so:

```
BYPASS_TETHYS_HOME_PAGE = True
```

1.7.4 Developer Tools

Tethys provides a Developer Tools page that is accessible when you run Tethys in developer mode. Developer Tools contain documentation, code examples, and live demos of the features of various features of Tethys. Use it to learn how to add a map or a plot to your web app using Gizmos or browse the available geoprocessing capabilities and formulate geoprocessing requests interactively.

****Figure 4.** ** Use the Developer Tools page to assist you in development.



Developer Tools



Gizmos

Gizmos are building blocks that can be used to create beautiful interactive controls in Tethys Apps. Using gizmos, developers can add date-pickers, plots, and maps to their templates with minimal coding. Follow the link to learn more.

[Show me the docs.](#)



Web Processing Services

Geoprocessing in Tethys apps can be accomplished using any web processing service (WPS). For convenience, Tethys provides the 52 North WPS service built in. Use this tool to explore the processes that are available and how to parameterize them.

[Go to tool.](#)

1.7.5 Manage Tethys Services

The administrator pages provide a simple mechanism for linking to the other services of Tethys Platform. Use the `Spatial Dataset Services` link to connect your Tethys Portal to GeoServer, the `Dataset Services` link to connect to CKAN instances or HydroShare, or the `Web Processing Services` link to connect to WPS instances. For detailed instructions on how to perform each of these tasks, refer to the *Spatial Dataset Services API*, *Dataset Services API*, and *Web Processing Services API* documentation, respectively.

1.7.6 Manage Computing Resources

Manage the computing resources of Tethys Portal using the `Tethys Compute` admin pages. Powered, by TethysCluster, these pages allow Tethys Portal administrators to spin up clusters of computing resources on either the Amazon or Microsoft Azure commercial clouds. These computational nodes will be made available to apps that are installed in your Tethys Portal. For more detailed information on the computing capabilities of Tethys Platform, refer to the *Distributed Computing API* documentaiton.

1.8 Deploy

Last Updated: May 23, 2015

The following instructions can be used to deploy Tethys Platform.

1.8.1 System Requirements

Last Updated: April 18, 2015

Tethys Platform is composed of several software components, each of which has the potential of using a copious amount of computing resources (see Figure 1). We recommend distributing the software components across several servers to optimize the use of computing resources and improve performance of Tethys Platform. Specifically, we recommend having a separate server for each of the following components:

- Tethys Portal
- PostgreSQL with PostGIS
- GeoServer
- 52 North WPS
- HTCondor
- CKAN

The following requirements should be interpreted as minimum guidelines. It is likely you will need to expand storage, RAM, or processors as you add more apps. Each instance of Tethys Platform will need to be fine tuned depending to fit the requirements of the apps that it is serving.

Tethys Portal

Tethys Portal is a Django web application. It needs to be able to handle requests from many users meaning it will need processors and memory. Apps should be designed to offload data storage onto one of the data storage nodes (CKAN, database, GeoServer) to prevent the Tethys Portal server from get bogged down with file reads and writes.

- Processor: 2 CPU Cores @ 2 GHz each

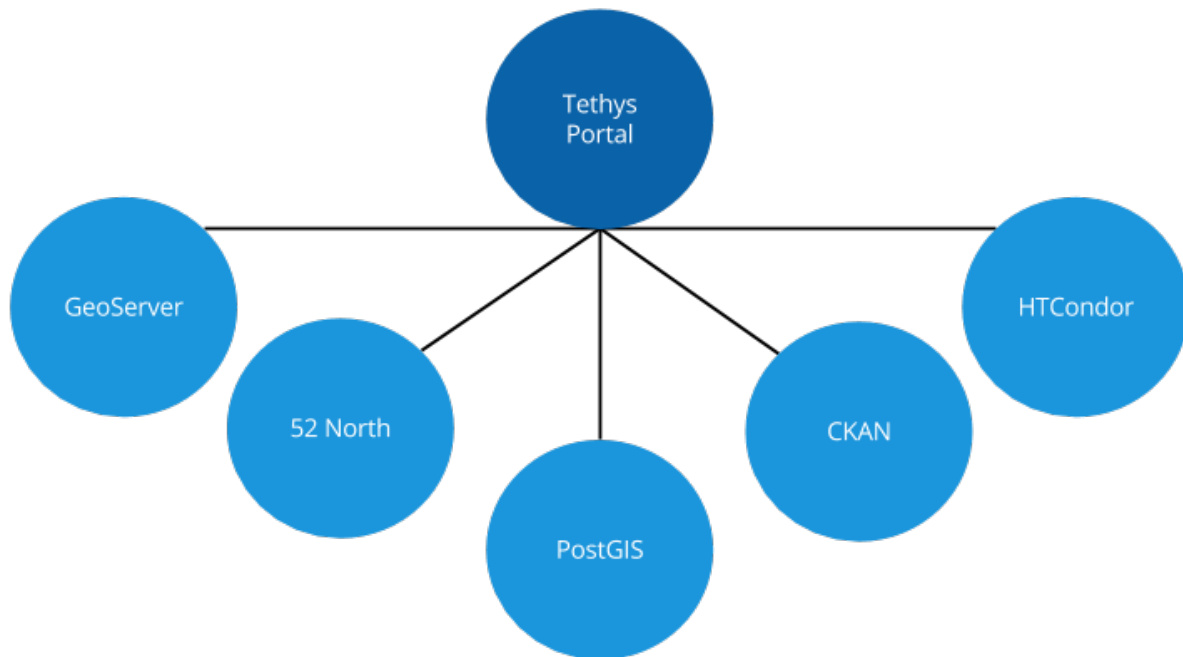


Figure 1.11: **Figure 1.** Tethys Platform consists of several software components that should be hosted on separate servers in a production environment.

- RAM: 4 GB
- Hard Disk: 20 GB

GeoServer

GeoServer is used to render maps and spatial data. It performs operations like coordinate transformations and format conversions on the fly, so it needs a decent amount of processing power and RAM. It also requires storage for the datasets that it is serving.

- Processor: 4 CPU Cores @ 2 GHz each
- RAM: 8 GB
- Hard Disk: 500 GB +

52 North WPS

52 North WPS is a geoprocessing service provider and as such will require processing power.

- Processors: 4 CPU Cores @ 2 GHz each
- RAM: 8 GB
- Hard Disk: 100 GB

PostgreSQL with PostGIS

PostgreSQL is a database server and it should be optimized for storage. The PostGIS extension also provide the server with geoprocessing capabilities, which may require more processing power than recommended here.

- Processors: 4 CPU Cores @ 2 GHz each
- RAM: 4 GB
- Hard Disk: 500 GB +

1.8.2 Production Installation

Last Updated: May 26, 2015

This article will provide an overview of how to install Tethys Portal in a production setup ready to host apps. The recommended deployment platform for Python web projects is to use [WSGI](#). The easiest and most stable way to deploy a WSGI application is with the [modwsgi](#) extension for the [Apache Server](#). These instructions are optimized for Ubuntu 14.04 using Apache and modwsgi, though installation on other Linux distributions will be similar.

1. Install Tethys Portal

Follow the default *Installation on Linux* instructions to install Tethys Portal with the following considerations

- Assign strong passwords to the database users.
- Create a new settings file, do not use the same file that you have been using in development.
- Optionally, Follow the *Distributed Configuration* instructions to install Docker and the components of the software suite on separate servers.

2. Install Apache and Dependencies

Install Apache and the `modwsgi` module if they are not installed already. In this tutorial, `vim` is used to edit file, however, you are welcome to use any text editor you are comfortable with.

```
$ sudo apt-get install apache2 libapache2-mod-wsgi vim
```

3. Make BASELINE Virtual Environment

An additional virtual environment needs to be created to use `modwsgi` in Apache. This virtual environment needs to be independent of the Tethys virtual environment and the system Python installation.

```
$ sudo mkdir -p /usr/local/pythonenv  
$ sudo virtualenv --no-site-packages /usr/local/pythonenv/BASELINE
```

4. Set WSGI Python Home

Edit the Apache configuration to use the `BASELINE` environment as the home python for WSGI. Open `apache2.conf` using `vim` or another text editor:

```
$ sudo vim /etc/apache2/apache2.conf
```

To edit the file using `vim`, you need to be in `INSERT` mode. Press `i` to enter `INSERT` mode and add this line to the bottom of the `apache2.conf` file:

```
WSGIPythonHome /usr/local/pythonenv/BASELINE
```

Press `ESC` to exit `INSERT` mode and then press `:x` and `ENTER` to save changes and exit.

5. Make Directories for Static Files and TethysCluster

When running Tethys Platform in development mode, the static files are automatically served by the development server. In a production environment the static files will need to be collected into one location and Apache or another server will need to be configured to serve these files (see [Deployment Checklist: STATIC_ROOT](#)). Since Apache will be serving Tethys Portal under Apache user (`www-data`) the TethysCluster home directory also needs to be created:

```
$ sudo mkdir /var/www/.tethyscluster && sudo mkdir -p /var/www/tethys/static
$ sudo chown `whoami` /var/www/tethys/static
```

6. Setup Email Capabilities

Tethys Platform provides a mechanism for resetting forgotten passwords that requires email capabilities, for which we recommend using Postfix. Install Postfix as follows:

```
$ sudo apt-get install postfix
```

When prompted select “Internet Site”. You will then be prompted to enter your Fully Qualified Domain Name (FQDN) for your server. This is the domain name of the server you are installing Tethys Platform on. For example:

```
foo.example.org
```

Next, configure Postfix by opening its configuration file:

```
$ sudo vim /etc/postfix/main.cf
```

Press `i` to start editing, find the `myhostname` parameter, and change it to point at your FQDN:

```
myhostname = foo.example.org
```

Find the `mynetworks` parameter and verify that it is set as follows:

```
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128
```

Press `ESC` to exit `INSERT` mode and then press `:x` and `ENTER` to save changes and exit. Finally, restart the Postfix service to apply the changes:

```
$ sudo service postfix restart
```

Django must be configured to use the postfix server. The next section will describe the Django settings that must be configured for the email server to work. For an excellent guide on setting up Postfix on Ubuntu, refer to [How To Install and Setup Postfix on Ubuntu 14.04](#).

7. Set Secure Settings

Several settings need to be modified in the `settings.py` module to make the installation ready for a production environment. The internet is a hostile environment and you need to take every precaution to make sure your Tethys Platform installation is secure. Django provides a [Deployment Checklist](#) that points out critical settings. You should review this checklist carefully before launching your site. As a minimum do the following:

Open the `settings.py` module for editing using `vim` or another text editor:

```
$ sudo vim /usr/lib/tethys/src/tethys_apps/settings.py
```

Press `i` to start editing and change the following settings:

1. Create new secret key

Create a new `SECRET_KEY` for the production installation of Tethys Platform. Do not use the same key you used during development and keep the key a secret. Take care not to store the `settings.py` file with the production secret key in a repository. Django outlines several suggestions for making the secret key more secure in the [Deployment Checklist: SECRET_KEY](#) documentation.

2. Turn off debugging

Turn off the debugging settings by changing `DEBUG` and `TEMPLATE_DEBUG` to `False`. **You must never turn on debugging in a production environment.**

```
DEBUG = False
TEMPLATE_DEBUG = False
```

3. Set the allowed hosts

Allowed hosts must be set to a suitable value, usually a list of the names and aliases of the server that you are hosting Tethys Portal on (e.g.: “`www.example.com`”). Django will not work without a value set for the `ALLOWED_HOSTS` parameter when debugging is turned of. See the [Deployment Checklist: ALLOWED_HOSTS](#) for more information.

```
ALLOWED_HOSTS = ['www.example.com']
```

4. Set the static root directory

You must set the `STATIC_ROOT` settings to tell Django where to collect all of the static files. Set this setting to the directory that was created in the previous step (`/var/www/tethys/static`). See the [Deployment Checklist: STATIC_ROOT](#) for more details.

```
STATIC_ROOT = '/var/www/tethys/static'
```

5. Set email settings

Several email settings need to be configured for the forget password functionality to work properly. The following example illustrates how to setup email using the Postfix installation from above:

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'localhost'
EMAIL_PORT = 25
EMAIL_HOST_USER = ''
EMAIL_HOST_PASSWORD = ''
EMAIL_USE_TLS = False
DEFAULT_FROM_EMAIL = 'Example <noreply@example.com>'
```

For more information about setting up email capabilities for Tethys Platform, refer to the [Sending email](#) documentation.

Press `ESC` to exit `INSERT` mode and then press `:x` and `ENTER` to save changes and exit.

Important: Review the [Deployment Checklist](#) carefully.

8. Create Apache Site Configuration File

Create an Apache configuration for your Tethys Platform using the **gen** command and open the `tethys-default.conf` file that was generated using `vim`:

```
$ sudo su
$ . /usr/lib/tethys/bin/activate
(tethys) $ tethys gen apache -d /etc/apache2/sites-available
(tethys) $ vim /etc/apache2/sites-available/tethys-default.conf
(tethys) $ exit
```

Press **i** to enter INSERT mode and edit the file. Copy and paste the following changing the `ServerName` and `ServerAlias` appropriately. The `tethys-default.conf` will look similar to this when you are done:

```
<VirtualHost 0.0.0.0:80>
    ServerName example.net
    ServerAlias www.example.net

    Alias /static/ /var/www/tethys/static/

    <Directory /var/www/tethys/static/>
        Require all granted
    </Directory>

    WSGIScriptAlias / /usr/lib/tethys/src/tethys_portal/wsgi.py

    <Directory /usr/lib/tethys/src/tethys_portal>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>

    # Daemon config
    WSGIDaemonProcess tethys_default \
        python-path=/usr/lib/tethys/src/tethys_portal:/usr/lib/tethys/lib/python2.7/
        site-packages
    WSGIProcessGroup tethys_default
    # Logs
    ErrorLog /var/log/apache2/tethys_default.error.log
    CustomLog /var/log/apache2/tethys_default.custom.log combined
</VirtualHost>
```

There is a lot going on in this file, for more information about Django and WSGI review Django's [How to deploy with WSGI](#) documentation.

9. Install Apps

Download and install any apps that you want to host using this installation of Tethys Platform. It is recommended that you create a directory to store the source code for all of the apps that you install. The installation of each app may vary, but generally, an app can be installed as follows:

```
$ sudo su
$ . /usr/lib/tethys/bin/activate
(tethys) $ cd /path/to/tethysapp-my_first_app
(tethys) $ python setup.py install
(tethys) $ exit
```

10. Run Collectstatic

The static files need to be collected into the directory that you created. Enter the following commands and enter “yes” if prompted:

```
$ sudo su
$ . /usr/lib/tethys/bin/activate
(tethys) $ tethys manage collectstatic
(tethys) $ exit
```

11. Setup the Persistent Stores for Apps

After all the apps have been successfully installed, you will need to initialize the persistent stores for the apps:

```
$ . /usr/lib/tethys/bin/activate
(tethys) $ tethys syncstores all
```

12. Change Ownership

When you are finished installing Tethys Portal, change the ownership of the source code and static files to be the Apache user (`www-data`):

```
$ sudo chown -R www-data:www-data /usr/lib/tethys/src /var/www/tethys/static
/var/www/.tethyscluster
```

13. Enable Site and Restart Apache

Finally, you need to disable the default apache site, enable the Tethys Portal site, and reload Apache:

```
$ sudo a2dissite 000-default.conf && sudo a2ensite tethys-default.conf &&
sudo service apache2 reload
```

Tip: To install additional apps after the initial setup of Tethys, you will follow the following process:

1. Change ownership of the `src` and `static` directories to your user using the patten in step 12 OR login as root user using `sudo su`.
 2. Install apps, syncstores, and collectstatic as in steps 9-11.
 3. Set the apache user as owner of `src` and `static` again as in 12.
 4. Reload the apache server using `sudo service apache2 reload`.
-

1.8.3 Distributed Configuration

Last Updated: April 18, 2015

The Tethys Docker images can be used to easily install each of the software components of Tethys Platform on separate servers. However, you will not be able to use the Tethys commandline tools to install the Dockers as you do during development. The following article describes how to deploy each software component using the native Docker API.

Install Docker on Each Server

After you have provisioned servers for each of the Tethys software components, install Docker on each using the appropriate [Docker installation instructions](#). Docker provides installation instructions for most major types of servers.

GeoServer Docker Deployment

Pull the Docker image for GeoServer using the following command:

```
$ sudo docker pull ciwater/geoserver
```

After the image has been pulled, run a new Docker container as follows:

```
$ sudo docker run -d -p 80:8080 --restart=always --name geoserver ciwater/geoserver
```

Refer to the [Docker Run Reference](#) for an explanation of each parameter. To summarize, this will start the container as a background process on port 80, with the restart policy set to always restart the container after a system reboot, and with an appropriate name.

More information about the GeoServer Docker can be found on the Docker Registry:

<https://registry.hub.docker.com/u/ciwater/geoserver/>

Important: The admin username and password can only be changed using the web admin interface. Be sure to log into GeoServer and change the admin password using the web interface. The default username and password are *admin* and *geoserver*, respectively.

Backup

PostgreSQL with PostGIS Docker Deployment

Pull the Docker image for PostgreSQL with PostGIS using the following command:

```
$ sudo docker pull ciwater/postgis
```

The PostgreSQL with PostGIS Docker automatically initializes with the three database users that are needed for Tethys Platform:

- tethys_default
- tethys_db_manager
- tethys_super

The default password for each is “pass”. For production, you will obviously want to change these passwords. Do so using the appropriate environmental variable:

- -e TETHYS_DEFAULT_PASS=<TETHYS_DEFAULT_PASS>
- -e TETHYS_DB_MANAGER_PASS=<TETHYS_DB_MANAGER_PASS>
- -e TETHYS_SUPER_PASS=<TETHYS_SUPER_PASS>

Here is an example of how to use the environmental variables to set passwords when starting a container:

```
$ sudo docker run -d -p 80:5432 -e TETHYS_DEFAULT_PASS="pass" -e  
TETHYS_DB_MANAGER_PASS="pass" -e TETHYS_SUPER_PASS="pass"
```

Refer to the [Docker Run Reference](#) for an explanation of each parameter. To summarize, this will start the container as a background process on port 80, with the restart policy set to always restart the container after a system reboot, and with an appropriate name. It also set the passwords for each database at startup.

More information about the PostgreSQL with PostGIS Docker can be found on the Docker Registry:

<https://registry.hub.docker.com/u/ciwater/postgis/>

Important: Set strong passwords for each database user for a production system.

Backup

52 North WPS Docker Deployment

Pull the Docker image for 52 North WPS using the following command:

```
$ sudo docker pull ciwater/n52wps
```

After the image has been pulled, run a new Docker container as follows:

```
$ sudo docker run -d -p 80:8080 -e USERNAME="foo" -e PASSWORD="bar" --restart=always  
--name n52wps ciwater/n52wps
```

Refer to the [Docker Run Reference](#) for an explanation of each parameter. To summarize, this will start the container as a background process on port 80, with the restart policy set to always restart the container after a system reboot, and with an appropriate name. It also sets the username and password for the admin user.

You may pass several environmental variables to set the service metadata and the admin username and password:

- -e USERNAME=<ADMIN_USERNAME>
- -e PASSWORD=<ADMIN_PASSWORD>
- -e NAME=<INDIVIDUAL_NAME>
- -e POSITION=<POSITION_NAME>
- -e PHONE=<VOICE>
- -e FAX=<FACSIMILE>
- -e ADDRESS=<DELIVERY_POINT>
- -e CITY=<CITY>
- -e STATE=<ADMINISTRATIVE_AREA>
- -e POSTAL_CODE=<POSTAL_CODE>
- -e COUNTRY=<COUNTRY>
- -e EMAIL=<ELECTRONIC_MAIL_ADDRESS>

Here is an example of how to use the environmental variables to set metadata when starting a container:

```
$ sudo docker run -d -p 80:8080 -e USERNAME="foo" -e PASSWORD="bar" -e NAME="Roger" -e COUNTRY="USA"
```

More information about the 52 North WPS Docker can be found on the Docker Registry:

<https://registry.hub.docker.com/u/ciwater/n52wps/>

Important: Set strong passwords for the admin user for a production system.

Maintaining Docker Containers

This section briefly describes some of the common maintenance tasks for Docker containers. Refer to the [Docker Documentation](#) for a full description of Docker.

Status

You can view the status of containers using the following commands:

```
# Running containers
$ sudo docker ps

# All containers
$ sudo docker ps -a
```

Start and Stop

Docker containers can be stopped and started using the names assigned to them. For example, to stop and start a Docker named “postgis”:

```
$ sudo docker stop postgis
$ sudo docker start postgis
```

Attach to Container

You can attach to running containers to give you a command prompt to the container. This is useful for checking logs or modifying configuration of the Docker manually. For example, to attach to a container named “postgis”:

```
$ sudo docker exec --rm -it postgis bash
```

1.8.4 Update from 1.0 to 1.1

Last Updated: April 24, 2015

The following article describes how to update your Tethys Platform installation from version 1.0.X to 1.1.0. There are two methods that can be used to update to 1.1.0:

1. Delete Tethys Virtual Environment

Use the following command to delete the Tethys Platform virtual environment:

```
$ sudo rm -rf /usr/lib/tethys
```

2. Reinstall Tethys Platform for Production

Follow the *Production Installation* instructions to reinstall Tethys Platform for production. Do not update the Docker images unless you want to loose all data in databases.

Warning: Updating the Docker containers will result in the loss of all data.

3. Reinstall Apps

Reinstall any apps that you wish to have installed on Tethys Platform. Refer to the documentation for each app for specific installation instructions, but generally apps can be installed as follows:

```
    $ . /usr/lib/tethys/bin/activate
(tethys) $ cd /path/to/tethysapp-my_first_app
(tethys) $ python setup.py install
(tethys) $ tethys syncstores my_first_app
```

The databases for the apps should have been retained unless you updated the Docker containers.

1.9 Source Code

Last Updated: May 23, 2015

The source code for Tethys Platform is contained in the following repositories:

- Tethys Platform
- Tethys Dockers
- Tethys Dataset Services
- TethysCluster
- CondorPy

1.10 Develop Tethys Platform

Last Updated: May 23, 2015

This section provides instructions for those wishing to contribute to Tethys Platform development.

Caution: This documentation is meant for those wishing to develop Tethys Platform—not for those wishing to develop Tethys Apps. If you want to develop Tethys Apps, please visit the normal installation instructions found here: *Installation*.

1.10.1 Development Installation

Last Updated: May 28, 2015

This section describes how to install Tethys Platform for development. Installation instructions are provided for Linux (Ubuntu).

Development Installation on Linux

Last Updated: May 28, 2015

Important: Tethys Platform is built on the Python web framework Django. If you are not familiar with Django development, it is required that you read and complete all of the tutorials in the **First Steps** section of the [Django Documentation](#) before embarking on Tethys Platform development.

1. Install the Dependencies

Complete steps 1-3 from the normal *Installation on Linux* instructions.

2. Create Project Directory

Create a project directory for the Tethys source code in a convenient location. This will be the directory you will work out of. For this tutorial, the directory will be called `tethysdev` and it will be located in the home directory. Also, create a directory inside your project directory called `apps` for any Tethys app projects you work on during development of Tethys.

```
$ mkdir ~/tethysdev
$ cd ~/tethysdev
$ mkdir apps
```

3. Pull the Source Code

Tethys Platform is versioned with Git and hosted on GitHub. Change into the project directory you created in the previous step and execute the following commands to download the source code:

```
$ git clone https://github.com/CI-WATER/tethys.git
$ git clone https://github.com/CI-WATER/tethys_dataset_services.git
$ git clone https://github.com/CI-WATER/tethys_docker.git
```

When you are done, your project directory should have the following contents:

```
tethysdev/
|-- apps/
|-- tethys/
|-- tethys_dataset_services/
|-- tethys_docker/
```

A brief explanation of each directory is provided below. For more details about the organization of Tethys Platform source code, see [Tethys Development Overview](#).

- **apps**: a directory to that will contain Tethys app projects used for development purposes.
- **tethys**: The main Django site project for Tethys. Most of the source for tethys platform is contained in this project.
- **tethys_dataset_services**: A Python module providing an interface with CKAN, HydroShare, and GeoServer that was made a separate Python module to allow it's use by web services external to Tethys Platform installations.
- **tethys_docker**: The Dockerfiles that can be used to create the Tethys Docker images.

4. Create Virtual Environment and Install Tethys Platform

Many of the Tethys commandline utilities depend on Tethys being installed in a default location: `/usr/lib/tethys/src` for unix environments. This location is not ideal for development, so instead you will create a symbolic link from where you downloaded the source to the default location. The virtual environment will also be installed in the default location.

1. If you have an existing installation of Tethys in `/usr/lib/tethys`, you should delete it:

```
$ sudo rm -rf /usr/lib/tethys
```

2. Create a *Python virtual environment* and activate it:

```
$ sudo mkdir -p /usr/lib/tethys
$ sudo chown `whoami` /usr/lib/tethys
$ virtualenv --no-site-packages /usr/lib/tethys
$ . /usr/lib/tethys/bin/activate
```

3. Create a symbolic link from the tethys source code you downloaded to the default location:

```
$ ln -s ~/tethysdev/tethys /usr/lib/tethys/src
```

4. Install each of the projects that you downloaded in development mode. Development mode any changes you make to the source code to take effect immediately without requiring you to reinstall the project:

```
$ cd ~/tethysdev/tethys_dataset_services && python setup.py develop
```

5. Install the Python modules that Tethys requires:

```
$ pip install -r /usr/lib/tethys/src/requirements.txt
$ python /usr/lib/tethys/src/setup.py develop
```

6. Restart the Python virtual environment:

```
$ deactivate
$ . /usr/lib/tethys/bin/activate
```

5. Install Tethys Software Suite Using Docker

1. Initialize the Tethys Software Suite Docker containers with the default parameters:

```
$ tethys docker init -d
```

Here are the default passwords that you will need to know to finish setup:

PostGIS Database User Passwords:

- **tethys_default**: pass
- **tethys_db_manager**: pass
- **tethys_super**: pass

Geoserver Username and Password:

- **admin**: geoserver

52 North WPS Username and Password:

- **wps**: wps

2. Start the Docker containers:

```
$ tethys docker start
```

Note: Although each Docker container appears to start instantaneously, it may take several minutes for the started containers to be fully up and running.

6. Create Settings File and Configure Settings

Create a new settings file for your Tethys Platform installation using the **tethys** *Command Line Interface*. Execute the following command in the terminal:

```
$ tethys gen settings -d /usr/lib/tethys/src/tethys_apps
```

This will create a file called `settings.py` in the directory `/usr/lib/tethys/src/tethys_apps`. Because your source code has been symbolically linked to this location, the `settings.py` file will also be located in `~/tethysdev/tethys/tethys_apps`. There are a few settings that need to be configured in this file.

Open the `settings.py` file that you just created from your source code location (`~/tethysdev/tethys/tethys_apps/settings.py`) in a text editor and modify the following settings appropriately.

1. Run the following command to obtain the host and port for Docker running the database (PostGIS). You will need these in the following steps:

```
$ tethys docker ip
```

2. Replace the password for the main Tethys Portal database, **tethys_default**, with the password you created in the previous step. Also make sure that the host and port match those given from the `tethys docker ip` command (PostGIS). This is done by changing the values of the `PASSWORD`, `HOST`, and `PORT` parameters of the `DATABASES` setting:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'tethys_default',
        'USER': 'tethys_default',
        'PASSWORD': 'pass',
        'HOST': 'localhost',
        'PORT': '5435'
    }
}
```

3. Find the `TETHYS_DATABASES` setting near the bottom of the file and set the `PASSWORD` parameters with the passwords that you created in the previous step. If necessary, also change the `HOST` and `PORT` to match the host and port given by the `tethys docker ip` command for the database (PostGIS):

```
TETHYS_DATABASES = {
    'tethys_db_manager': {
        'NAME': 'tethys_db_manager',
        'USER': 'tethys_db_manager',
        'PASSWORD': 'pass',
        'HOST': '127.0.0.1',
        'PORT': '5435'
    },
    'tethys_super': {
        'NAME': 'tethys_super',
        'USER': 'tethys_super',
        'PASSWORD': 'pass',
        'HOST': '127.0.0.1',
        'PORT': '5435'
    }
}
```

4. Save your changes and close the `settings.py` file.

7. Create Database Tables

Execute the **tethys manage syncdb** command from the Tethys *Command Line Interface* to create the database tables. In the terminal:

```
$ tethys manage syncdb
```

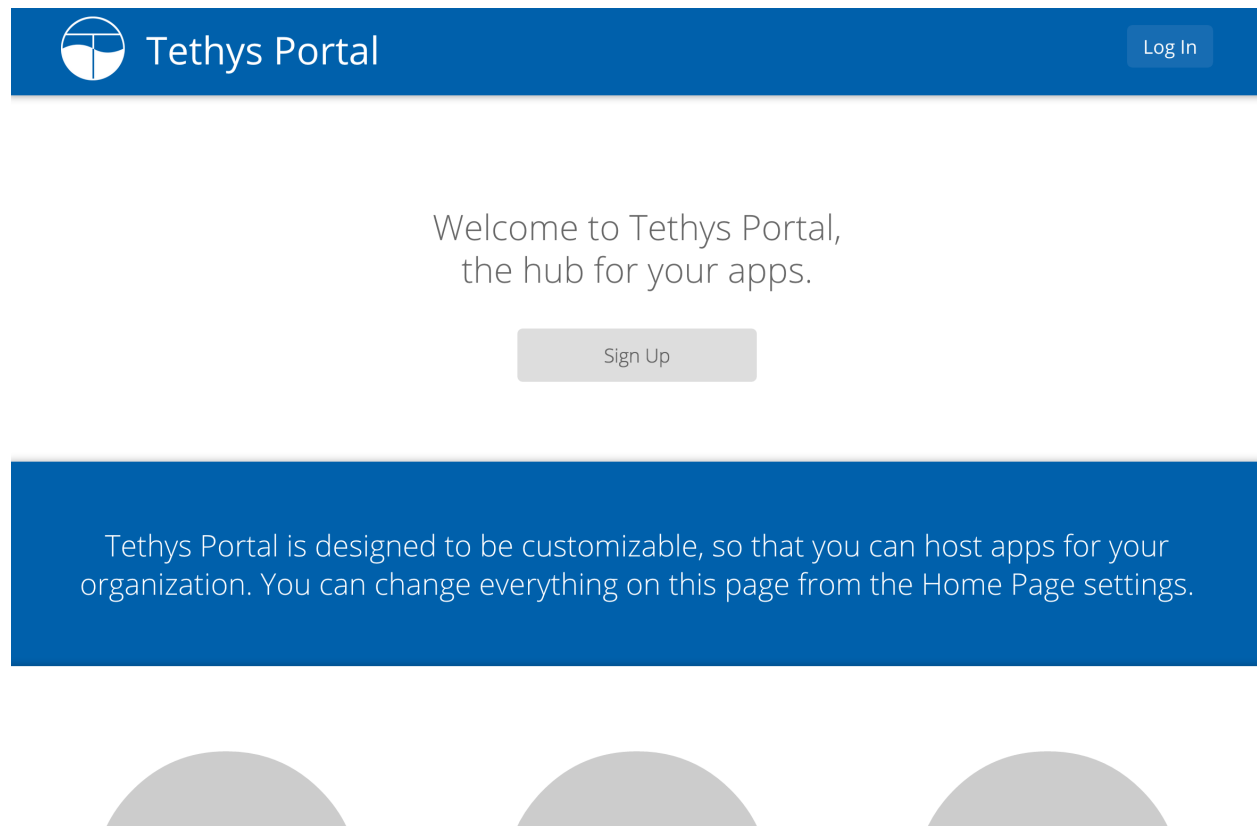
Important: When prompted to create a system administrator enter ‘yes’. Take note of the username and password, as this will be the user you use to manage your Tethys Portal.

8. Start up the Django Development Server

You are now ready to start the development server and view your instance of Tethys Platform. The website that is provided with Tethys Platform is called Tethys Portal. In the terminal, execute the following command from the Tethys *Command Line Interface*:

```
$ tethys manage start
```

Open <http://localhost:8000/> in a new tab in your web browser and you should see the default Tethys Portal landing page.



9. Web Admin Setup

You are now ready to configure your Tethys Platform installation using the web admin interface. Follow the *Web Admin Setup* tutorial to finish setting up your Tethys Platform.

1.10.2 Tethys Development Overview

Last Updated: March 3, 2015

Warning: UNDER CONSTRUCTION

Django Project

Modular Design

Git Repositories

Branches

1.10.3 Docker Development

Last Updated: March 3, 2015

Warning: UNDER CONSTRUCTION

1.11 Supplementary

Last Updated: May 27, 2015

This section provides a list of miscellaneous reference material that can be used to help you understand Tethys Platform and Tethys app development in more detail.

1.11.1 Key Concepts

Last Updated: April 6, 2015

The purpose of this page is to provide an explanation of some of the key concepts of Tethys Platform. The concepts are only discussed briefly here to provide a basic overview. It is highly recommended that you visit the suggested resources to have a better understanding of these concepts, as developing apps in Tethys Platform relies heavily on them.

What is an App?

In the most basic sense, an app is a workflow. The purpose of an app is not provide an all-in-one solution, but rather to perform a narrowly focused task or set of tasks. For example, an app that works with hydrologic models might be focused on guiding the user to change the land use layer of a model, run the modified model, and compare the result with the original model results.

In terms of implementation, an app built with Tethys Platform or a Tethys app is a web app(as opposed to a mobile app). A Tethys Platform installation provides a website called the Tethys Portal that can be used to organize and access your apps. Tethys apps are technically extensions of the Tethys Portal web page, because when you create a Tethys app you will be adding additional web pages to the Tethys Portal web site. Tethys Platform is built on the Django Python web framework, so Tethys apps are also Django web apps—though Tethys Platform streamlines many aspects of Django web development. This is why the Django documentation is referred to often in the documentation for Tethys Platform.

Web Frameworks

Tethys Portal is built using the [Django](#) web framework. Understanding the difference between a static website and a dynamic website built with a web framework is important for app developers, because apps rely on web framework concepts.

Static web development consists of creating a series of HTML files—one for every page of the website. The files are organized using the server's file system and stored in some directory on the server that is accessible by the Internet. For a static site, the URL works very similar to how a file path works on an operating system. When a request is sent from the web browser to a server, the server locates the HTML file that the URL is requesting and returns it to the browser for the user to view.

The static method of developing web pages presents some problems for developers. For example, if a developer wants to include a consistent header and footer on every page of her website, she would end up duplicating the header and footer code many times (via copy and paste). As a result, static websites are more difficult to update and maintain, because changes need to be made wherever the code is duplicated. Developing a website in this way is error prone and can become prohibitive for large websites.

Web frameworks provide a way to develop websites using a programmatic approach. Instead of static HTML files, developers create generic reusable HTML templates. With a web framework, the developer can create one template file containing only the markup for the header and another template file for the footer. Now when the developer wants to include the header and footer in another page, she uses an import construct that references the header and footer templates. The header and footer markup is added dynamically to all the files that need it upon request by a template rendering program. Maintenance is much easier, because changes to the header and footer only need to be made in one place and the entire site will be updated. In this way, the site becomes dynamic. One type of software that makes it possible to create dynamic web pages is a web framework.

Web frameworks also handle requests differently than traditional web pages. When the user submits a request to the server, instead of looking up a file on the server at the directory implied by the URL, the request is handed to the web framework application. The web framework application processes the request and usually returns a web page that has been generated dynamically as the result. This type of web framework application is called a [Common Gateway Interface \(CGI\)](#) application; or if the application is a Python web framework, it is called a [Web Server Gateway Interface \(WSGI\)](#) application.

Model View Controller

The dynamic templating feature is only one aspect of what web frameworks offer. Many web frameworks use a software development pattern called [Model View Controller \(MVC\)](#). MVC is used to organize the code that is used to develop user interfaces into conceptual components. A brief explanation of each component is provided:

Model The model represents the code that is used to store and retrieve data that is used in the web application. Most websites use SQL databases for persistent data storage, so the model is usually made up of a database model.

View Views are used to represent the data to the end user. In a web applications views are the HTML pages that are generated. Views are typically generic, reusable, and oblivious to the origins of the data that fills them. This is possible because of templating languages that allow coders to create dynamic HTML web pages.

Controller Controllers are used to orchestrate the interaction between the view and the model. They contain the logic for retrieving data from the database and transforming it into a format that is consumable by the view, because the model and the view never communicate directly. Controllers also handle the input from the user.

When a user submits a request, the web application (dispatcher in the Figure above) looks up the controller that is mapped to the URL and executes it. The controller may perform change or lookup data from the model after which it returns this data and a template to render. This is handed off to a template rendering utility that processes the template

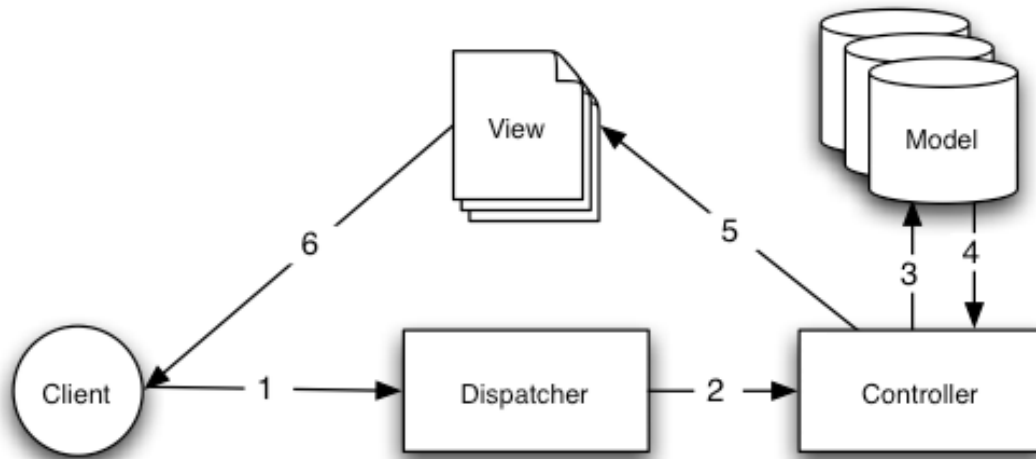


Figure 1.12: A typical collaboration of MVC components (courtesy of [Cake PHP Docs](#))

and generates HTML. The HTML is rendered for the user’s viewing pleasure in the web browser or other client. The user sends another request, and the process repeats.

URL Design and REST Paradigm

The URL takes on a different meaning in dynamic websites than it does in a static website. In a static website, the URL maps to directories and files on the server. In a dynamic website, there are no static files (or at least very few) to map to. The web framework simply maps the URL to a controller and returns the result. Although the developer is free to use URL’s in whatever manner they would like, it is recommended that some type of URL pattern should be used to make the website more maintainable.

We recommend developers use some form of the Representational State Transfer (REST) abstraction for creating meaningful URLs for apps. In a REST architecture for a website, the data of the website is referred to as resources. The current state of resources is presented to the user through some representation, for example, an HTML document. The user can interact with the resources through the actions of the controller. Examples of common actions on resources are create, read (view), update (edit), and delete, often referred to as CRUD. In true REST implementations, the CRUD operations are mapped to specific HTTP methods: POST, GET, PUT, and DELETE, respectively (see [HTTP Verbs](#)). In practice HTML only supports the POST and GET HTTP methods, so a pseudo-REST implementation is achieved via URL patterns.

For example, consider an app that is meant to provide information about a stream gages. In this case, the resources of the website may be stream gage records in a database. A potential URL for a page that shows a summary about a single stream gage record would be:

```
www.example.com/gages/1/show
```

The number “1” in the URL represents the stream gage record ID in the database. To show a page with the representation of another stream gage, the ID number could be changed. A generalization of this URL pattern could be represented as:

```
/gages/{id}/{action}
```

In this URL pattern, variables are represented using curly braces. The `{id}` variable in the URL represents the ID of a stream gage resource in our database and the `{action}` variable represents the action to perform on the stream gage resource. The `{action}` variable is used instead of HTTP methods to indicate which CRUD operation to perform

on the resource. In the first example, the action “show” is used to perform the read operation. Often, the show action is the default action, so the URL could be shortened to:

```
www.example.com/gages/1
```

Similarly, a URL for a page that represents all of the stream gages in the database in a list could be represented by omitting the ID:

```
www.example.com/gages
```

URLs for each of the CRUD operations on the stream gage database could look like this:

```
# Create
www.example.com/gages/new

# Read One
www.example.com/gages/1

# Read All
www.example.com/gages

# Update
www.example.com/gages/1/edit

# Delete
www.example.com/gages/1/delete
```

Before you dive into writing your app, you should take some time to design the URLs for the app. Define the resources for your app and the URLs that will be used to perform the CRUD operations on the resources.

Caution: The examples above used integer IDs for simplicity. However, using integer IDs in URLs is not recommended, because they are often incremented consecutively and can be easily guessed. For example, it would be very easy for an attacker to write a script that would increment through integer IDs and call the delete method on all your resources. A better option would be to use randomly assigned IDs such as a [UUID](#).

HTTP Verbs

Anytime you type a URL into an address bar you are performing what is called a GET request. All of the above URLs are examples of implementing REST using only GET requests. GET is an example of an HTTP verb or method. There are quite a few HTTP verbs, but the other verbs pertinent to REST are POST, PUT, and DELETE. A truly RESTful design would make use of these HTTP verbs to implement the CRUD for the resources instead of using different key word actions. Consider our example from above. To read or view a dog resource, we use a GET request as before:

```
HTTP GET
www.example.com/dogs/1
```

However, to implement the create action for a dog resource, now we use the POST verb with the same url that we used for the read action:

```
HTTP POST
www.example.com/dogs/1
```

Similarly, to delete the dog resource we use the same URL as before but this time use the DELETE verb and to update or edit a dog resource, we use the PUT verb. Using this pattern, the URL becomes a unique resource identifier (URI) and the HTTP verbs dictate what action we will perform on the data. Unfortunately, HTML (which is the interface of HTTP) does not implement PUT or DELETE verbs in forms. In practice many RESTful sites use the “action” pattern for interacting with resources, because not all of the HTTP verbs are supported.

1.11.2 App Project Structure

Last Updated: November 17, 2014

The source code for a Tethys app project is organized in a specific file structure. Figure 1 illustrates the key components of a Tethys app project called “my_first_app”. The top level package is called the *release package* and it contains the *app package* for the app and other files that are needed to distribute the app. The key components of the *release package* and the *app package* will be explained in this article.

Release Package

As the name suggests, the release package is the package that you will use to release and develop your app. The entire *release package* should be provided when you share your app with others.

The name of a *release package* follows a specific naming convention. The name of the directory should always start with “tethysapp-” followed by a *unique* name for the app. The name of the app may not have spaces, dashes, or other special characters (however, underscores are allowed). For example, Figure 1 shows the project structure for an app with name “my_first_app” and the name of the *release package* is “tethysapp-my_first_app”.

The release package must contain a setup script (`setup.py`) and `tethysapp` namespace package at a minimum. This directory would also be a good place to put any accessory files for the app such as a README file or LICENSE file. No code that is required by the app to run should be in this directory.

The setup script to install your app and its dependencies. A basic setup script is generated as part of the scaffolding for a new app project. For more information on writing setup scripts refer to the *Distributing Apps* tutorial and this article: [Writing the Setup Script](#).

The `tethysapp` package is a Python namespace package. It provides a way to mimic the production environment during development of the app (i.e.: when the app is installed, it will reside in a namespace package called `tethysapp`). This package contains the *app package*, which has the same name as your app name by convention.

Caution: When you generate a new app project using the command line tool, you will notice that many of the directories contain a `__init__.py` file, many of which are empty. These are omitted in the diagram for simplicity. **DO NOT DELETE THE `__init__.py` FILES.** These files indicate to Python that the directories containing them are Python packages. Your app will not work properly without the `__init__.py` files.

The App Package

The *app package* contains all of the source code and resources that are needed by the Tethys Platform to run your app. The `model.py`, `templates`, and `controllers.py` modules and directories correspond with the Model View Controller approach that is used to build apps.

The data structures, classes, and methods that are used to define the data model `model.py` module. The `templates` directory contains all the Django HTML templates that are used to generate the views of the app. The `controllers.py` module contains Python files for each controller of the app. The `public` directory is used for static resources such as images, JavaScript and CSS files. The `app.py` file contains all the configuration parameters for the app.

To learn how to work with the files in the *app package*, see the *Getting Started* tutorial.

Naming Conventions

There are a few naming conventions that need to be followed to avoid conflicts with other apps. The more obvious one is the *app package* name. Like all Python modules, *app package* names must be unique.

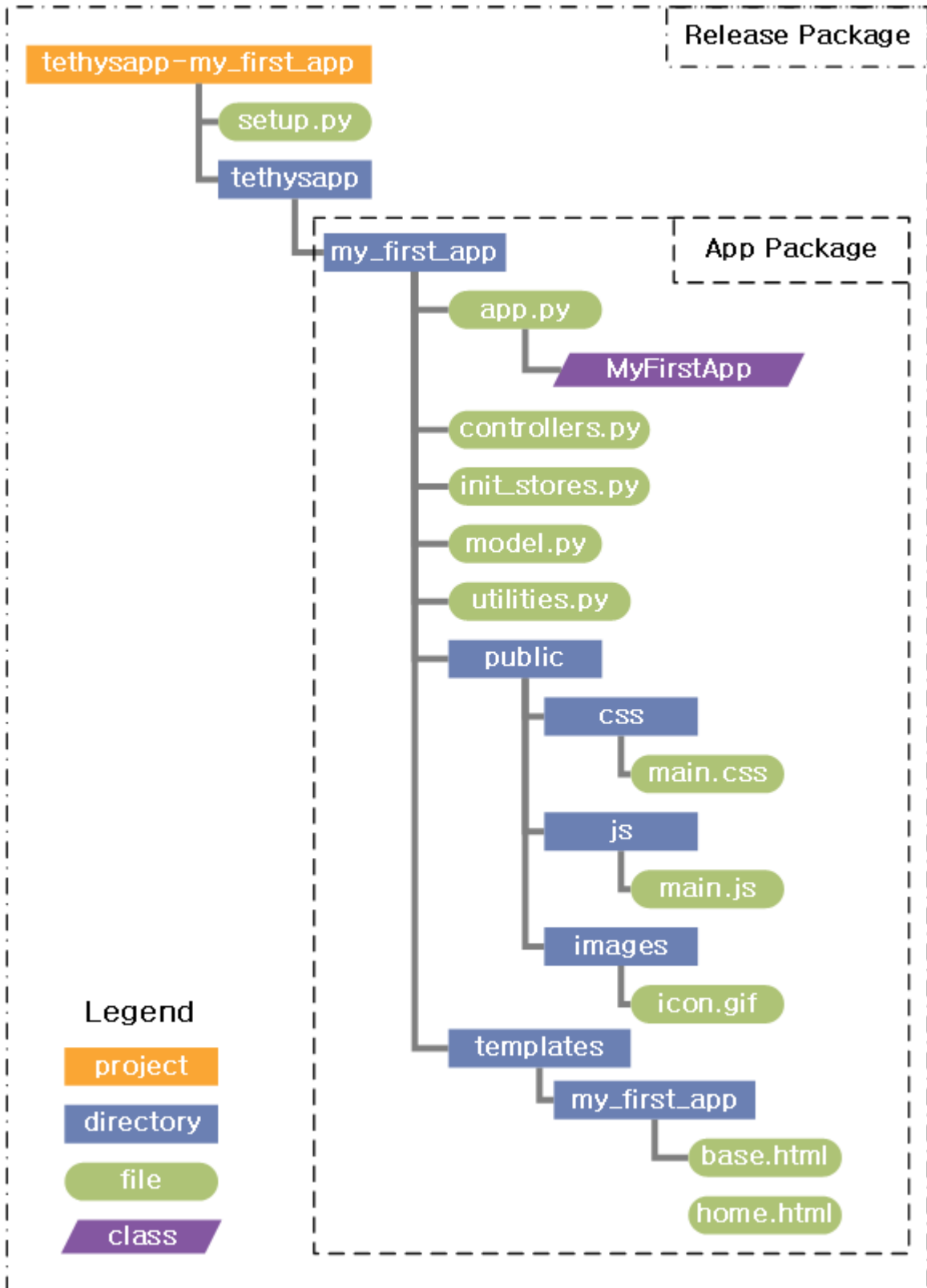


Figure 1.13: Figure 1. An example of a Tethys app project for an app named “my_first_app”.

All templates should be contained in a directory that shares the same name as the *app package* within the `templates` directory (see Figure 1). This ensures that when your app calls for a template like `home.html` it finds the correct one and not an `home.html` from another app.

1.11.3 Terminal Quick Guide

Last Updated: November 18, 2014

To install and use Tethys Platform, you will need to be familiar with using the command line/terminal. This guide provides tips and explanations of the most common features of command line that you will need to know to work with Tethys. For a more exhaustive reference, please review this excellent tutorial: [Learn the Bash Command Line](#).

\$

The “\$” in code blocks means “run this in the terminal”. This is usually done by typing the command or copying and pasting it into the terminal. When copying, don’t copy the “\$”. Copy lines one at a time and press `enter` after each one to execute it. Note that some commands may prompt you for input.

~

The “~” is short hand for your `Home` directory. You will see this symbol most often in paths that extend from your `Home` directory. The shorthand is used because the path to the `Home` directory varies depending on your user name. For example, if your user name was “john”, then the absolute path to your home directory would be something like `/home/john`.

sudo

Some operations on the commandline require authorization by a superuser or administrator. The `sudo` command is used to grant permission. This is done by prepending any command with `sudo`. You will be prompted for your password before you can continue.

```
sudo apt-get moo
```

Note: When you type passwords into the command line, the characters are not printed to the screen for security reasons. This can be unsettling, but type with faith and press `enter`.

cd

This command is used to change working directories on the command line. This is the equivalent of moving in and out of folders on a file browser.

mkdir

This command is used to make new directories.

chown

This command is used to change the owner of files or directories.

Copy and Paste

The keyboard shortcuts CTRL-C and CTRL-V do not do preform copy and paste in the terminal. Instead, use the shortcuts CTRL-SHIFT-C and CTRL-SHIFT-V to copy and paste.

1.11.4 Ubuntu Installation

Last Updated: November 17, 2014

Ubuntu Desktop can be downloaded at the [Download Ubuntu](#) page. There are three ways you can install Ubuntu on your computer. The first option is to overwrite whatever operating system you are running with Ubuntu. This can be done using either a USB or DVD. Use the [Install Ubuntu](#) instructions to do so (Note: these instructions are for Ubuntu 14.04, but they should work for Ubuntu 12.04 as well). This method is not usually preferable or recommended, because most users still want to retain use of their Windows or Mac operating systems. The next two options accomodate this need.

The second options is to install Ubuntu in a dual boot configuration. This will let you choose to either run Ubuntu or Windows/Mac OSX when you start your computer. Follow the instructions provided by Ubuntu for [Windows Dual Boot](#) if on a Windows computer or the [Intel Mac Dual Boot](#) if on a Mac computer.

The third option is to install Ubuntu as a virtual machine using virtualization software such as [VirtualBox](#). If you are running Mac OSX you can also use [VMWare](#) or [Parallels](#). Follow the instructions for creating a new Ubuntu virtual machine for the software you are running.

After installing Ubuntu, be sure to install any updates using the Update Manager and restart.

1.11.5 Test Docker Containers

If you would like, you may perform the following tests to ensure the containers are working properly.

Activate the virtual environment if you have not done so already and use the following Tethys command to start the Docker containers:

```
$ . /usr/lib/tethys/bin/activate
$ tethys docker start
```

Note: Although each Docker container seem to start instantaneously, it may take several minutes for the started containers to be fully up and running.

Use the following command in the terminal to obtain the ports that each software is running on:

```
$ tethys docker ip
```

You will be able to access each software on `localhost` at the appropriate port. For example, GeoServer and 52 North WPS both have web administrative interfaces. In a web browser, enter the following URLs replacing the `<port>` with the appropriate port number from the previous command:

```
# GeoServer
http://localhost:<port>/geoserver

# 52 North WPS
http://localhost:<port>/wps
```

With some luck, you should see the administrative page for each. Feel free to explore. You can login to the 52 North WPS admin site using the username and password you specified during installation or the defaults if you accepted those which are:

Default 52 North WPS Admin

- Username: wps
- Password: wps

You are not given the option of specifying a custom username and password for GeoServer, because it can only be done through the web interface. You may log into your GeoServer using the default username and password:

Default GeoServer Admin

- Username: admin
- Password: geoserver

The PostgreSQL database is installed with the database users and databases required by Tethys Platform: **tethys_default**, **tethys_db_manager**, and **tethys_super**. You set the passwords for each user during installation of the container. You can test the database by installing the [PGAdmin III](#) desktop client for PostgreSQL and using the credentials of the **tethys_super** database user to connect to it. For more detailed instructions on how to do this, see the [PGAdmin III Tutorial](#).

1.11.6 PGAdmin III Tutorial

Last Updated: November 20, 2014

All of the SQL databases used in Tethys Platform are [PostgreSQL](#) databases. An excellent graphical client for PostgreSQL. It is available for Windows, OSX, and many Linux distributions. Please visit the [Download](#) page to learn how to install it for your particular operating system. After it is installed, you can connect to your Tethys Platform databases by using the credentials for the `tethys_super` database user you defined during installation.

To create a new connection to your PostgreSQL database using PGAdmin:

1. Open PGAdmin III and click on the Add New Connection button.
2. In the New Server Registration dialog that appears, fill out the form with the appropriate credentials. Provide a meaningful name for the connection like “tethys”. If you have installed PostgreSQL with the Docker containers, the host will be either `localhost` if you are on Linux or `192.168.59.103` if you are on Mac or Windows. Use the `tethys docker ip` command to get the port for PostgreSQL (PostGIS). Fill in the username as `tethys_super` and enter the password you gave the user during installation. Click OK to close the window.
3. To connect to the PostgreSQL database server, double-click on the “tethys” connection listed under the Servers dropdown menu. You will see a list of the databases on the server. Expand the menus to view each database. The tables will be located under Schemas > public > Tables.

1.12 Summary of References

Last Updated: November 17, 2014

This page provides a list of all the external resources referred to during the documentation for convenience.

Ubuntu

- [Download Ubuntu](#)
- [Dual Boot Ubuntu with Windows PC](#)

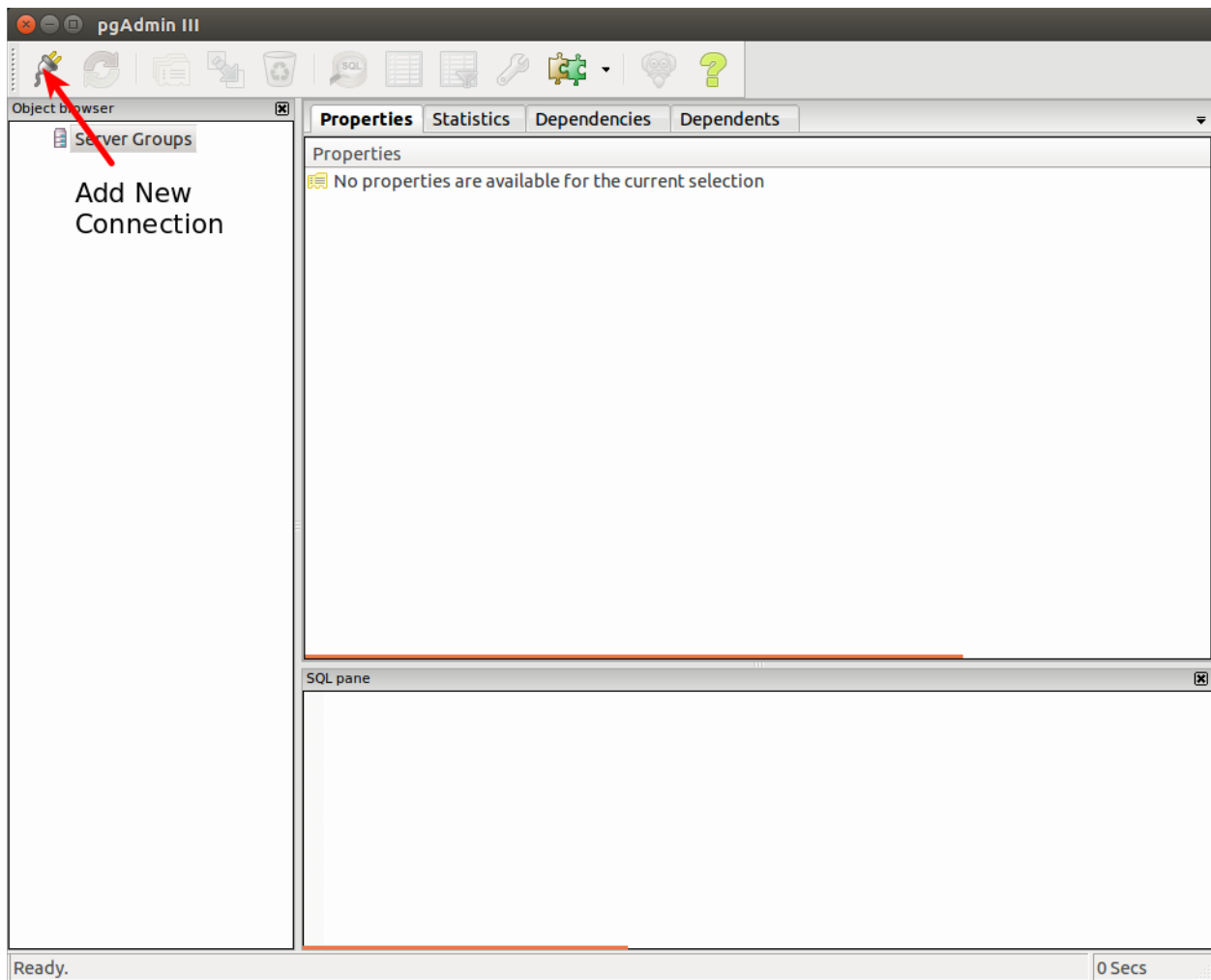


Figure 1.14: **Figure 1.** Click the Add New Connection button.

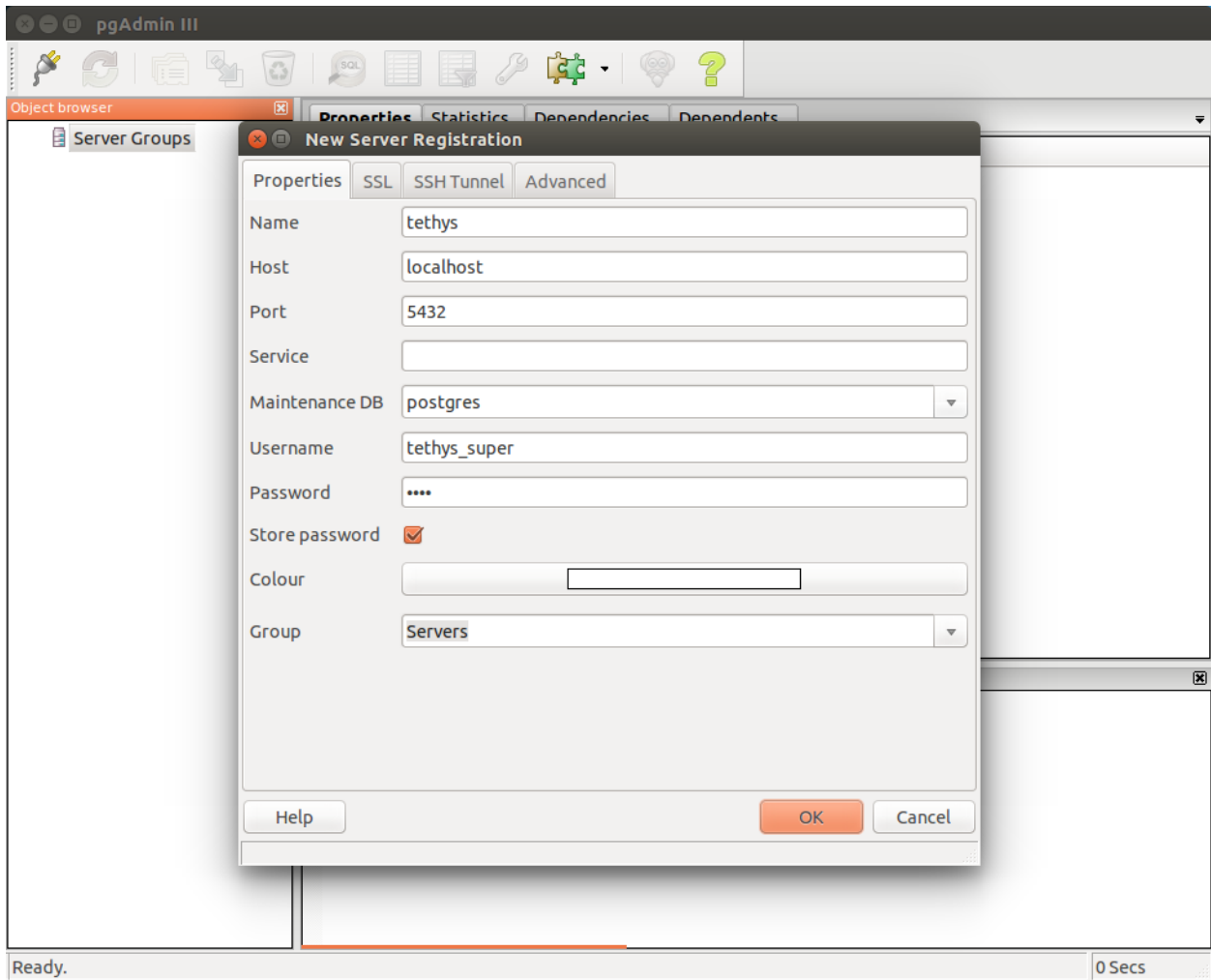


Figure 1.15: **Figure 2.** Fill out the New Server Registration dialog.

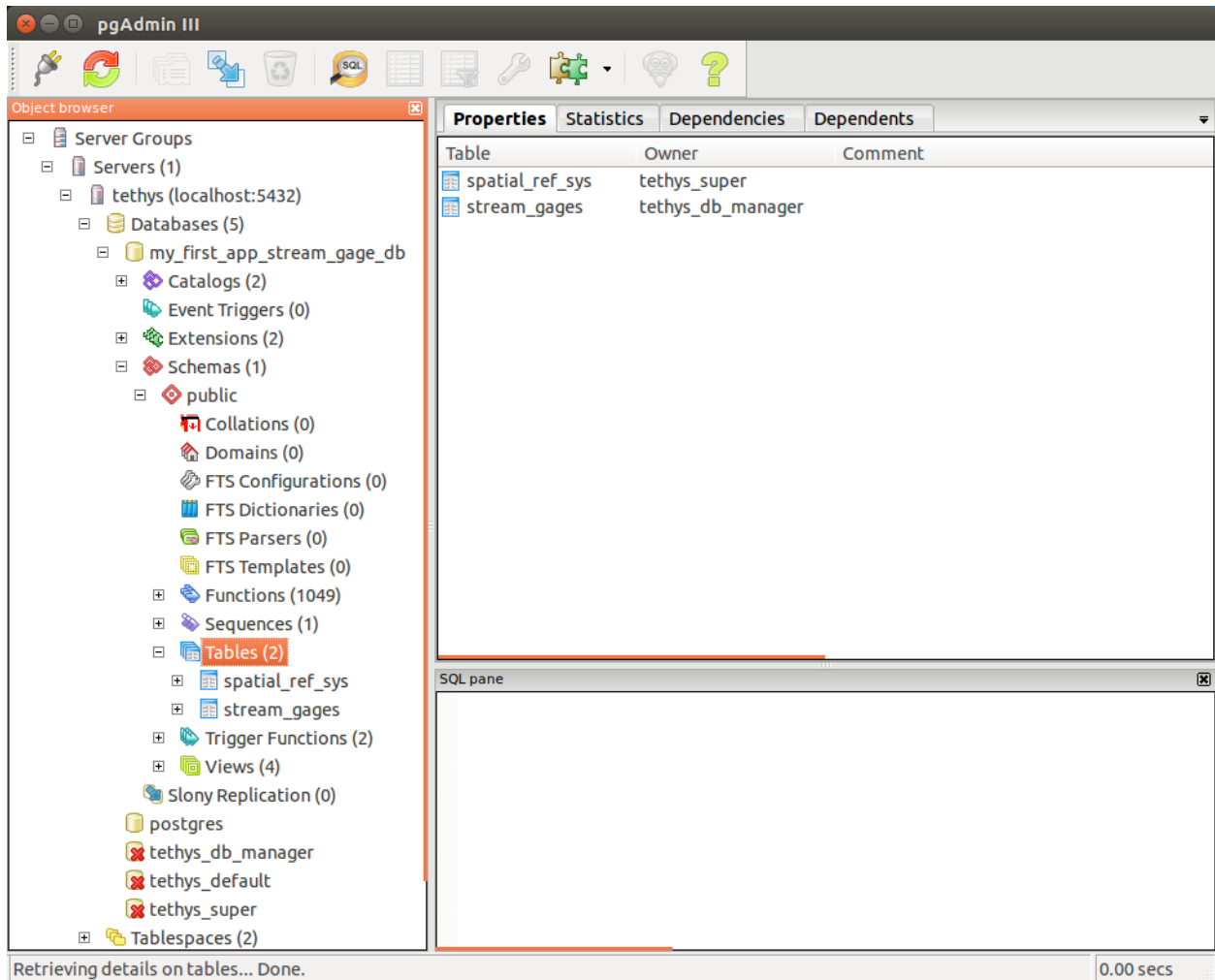


Figure 1.16: **Figure 3.** Browse the databases using the graphical interface.

- Learn the Bash Command Line

Docker

- Docker virtualization container

Virtualization Options

- VMWare
- Parallels
- VirtualBox

Django

- Writing Views
- Django Template Language
- Django template Variables
- Django Filter Reference
- Django Tag Reference
- Django Template Inheritance
- Django static tag
- Cross Site Forgery protection

CKAN

- Install CKAN 2.2 from Source
- CKAN instances around the world
- FileStore Setup
- DataStore Setup
- Install on Other Operating Systems
- Actions API

IDEs

- How to Install Aptana on Ubuntu 12.04

SQLAlchemy

- SQLAlchemy
- Object Relational Tutorial

GeoAlchemy

- GeoAlchemy2
- GeoAlchemy ORM
- Well Known Text

Database Clients

- PGAdmin III

PostGIS

- PostGIS

- [Geometry Function Reference](#)
- [Raster Function Reference](#)

Google

- [Obtaining an API Key](#)

Python

- [PyPI](#)
- [Setuptools Documentation](#)
- [Writing Setup Script](#)
- [Namespace](#)

Production Installation

- [WSGI](#)
- [modwsgi](#)
- [Deployment Checklist](#)
- [Deployment Checklist: STATIC_ROOT](#)
- [Deployment Checklist: SECRET_KEY](#)
- [Deployment Checklist: ALLOWED_HOSTS](#)
- [Deployment Checklist: STATIC_ROOT](#)
- [How to deploy with WSGI](#)
-

Miscellaneous

- [Universally unique identifier](#)
- [The Definitive Guide to GET vs POST](#)

1.13 Glossary

Last Updated: November 18, 2014

app class A class defined in the *app configuration file* that inherits from the `TethysAppBase` class provided by the Tethys Platform. For more details on the app class, see *App Base Class API*.

app configuration file A file located in the *app package* and called `app.py` by convention. This file contains the *app class* that is used to configure apps. For more details on the app configuration file, see *App Base Class API*.

app harvester An instance of the `SingletonAppHarvester` class. The app harvester collects information about each app and uses it to load Tethys apps.

app instance, app instances An instance of an *app class*.

app package, app packages A Python namespace package of a Tethys *app project* that contains all of the source code for an app. The app package is named the same as the app by convention. Refer to Figure 1 of *App Project Structure* for more information.

app project All of the source code for a Tethys app including the *release package* and the *app package*.

dataset, datasets A dataset is a container for one or more resources that are stored in a *dataset service*.

- dataset service, dataset services** A dataset service is a web service external to Tethys Platform that can be used to store and publish file-based datasets (e.g.: text files, Excel files, zip archives, other model files). See the *Dataset Services API* for more information.
- Debian** Debian is a type of Linux operating system and many Linux distributions are based on it including Ubuntu. See *Linux Distributions* for more information.
- Gizmo, Gizmos** Reusable view elements that can be inserted into a template using a single line of code. Examples include common GUI elements like buttons, toggle switches, and input fields as well as more complex elements like maps and plots. For more information on Gizmos, see *Template Gizmos API*.
- Model View Controller** The development pattern used to develop Tethys apps. The Model represents the data of the app, the View is composed of the representation of the data, and the Controller consists of the logic needed to prepare the data from the Model for the View and any other logic your app needs.
- persistent store, persistent stores** A persistent store is a database that can be automatically created for an app. See *The Model and Persistent Stores* tutorial and the *Persistent Stores API* for more information about persistent stores.
- release package** The top level Python namespace package of an *app project*. The release package contains the *setup script* and all the source for an app including the *app package*. Refer to Figure 1 of *App Project Structure* for more information.
- resource, resources** A resource is a file or other object and the associated metadata that is stored in a *dataset service*.
- setup script** A file located in the *release package* and called `setup.py` by convention. The setup script is used to automate the installation of apps. For more details see *Distributing Apps*.
- spatial dataset, spatial datasets** A spatial dataset is a file-based dataset that stores spatial data (e.g.: shapefiles, GeoTiff, ArcGrid, GRASS ASCII Grid).
- virtual environment, Python virtual environment** An isolated Python installation. Many operating systems use the system Python installation to perform maintenance operations. Installing Tethys Platform in a virtual environment prevents potential dependency conflicts.
- wps service, wps services** A WPS Service provides processes/geoprocesses as web services using the Open Geospatial Consortium Web Processing Service (WPS) standard.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1135482