|  |
| --- |
|  |
| New Datasets Documentation |
| Online decision support toolkit for climate resilient seaports |

|  |
| --- |
| **CLIMATE SMART SEAPORTS - PACIFIC** |
| New Datasets Documentation |
|  |

|  |
| --- |
| **RMIT University**  Prepared by: **Guillaume Prevost**  Date: **15/05/2014**  Document Version: **1.0** |

|  |  |
| --- | --- |
| **USAID Activity Number** | CCRDACD0005 |
| **Project Title** | Decision-support toolkit: towards Climate Smart Seaports in the Pacific Islands |
| **Organisation responsible for the project** | RMIT University |
| **Name of Contact Person** | Professor Darryn McEvoy |
| **Address and contact details of Contact Person** | Address: GPO Box 2476, Melbourne VIC 3001  Telephone: 03 99251943  Email: [Darryn.mcevoy@rmit.edu.au](mailto:Darryn.mcevoy@rmit.edu.au) |
| **Names and affiliations of collaborators if any** | Climate Change Adaptation Program and e-Research at RMIT University. |

# License

*Copyright (c) 2014, RMIT University, Australia.*

*All rights reserved.*

*This project’s code and documentation are under the BSD license. See 'license.txt' for details.*

*Project hosted at:*

[*https://bitbucket.org/guillaumeprevost/seaports-pacific.git*](https://bitbucket.org/guillaumeprevost/seaports-pacific.git)

Table of Contents

License 1

Document Revision History 3

1 Introduction 4

1.1 Project summary 4

1.2 Purpose of the document 4

1.3 Intended Audience and Reading Suggestions 4

1.4 References 4

2 Overview 5

3 Create a new Data Source 8

3.1 Create a Data Model (optional) 8

3.2 Create a Data Source class 9

3.3 Add Data Source definition in the database 11

3.4 Controller implementation (Optional) 12

3.5 Create Views for the display types 13

# Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Author | Date | Modifications | Version |
| Guillaume Prevost | 15/05/2014 |  | 1.0 |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Project summary

The potential impact of climate change on ports differs according to their location, function and business model of the ports. To be 'Climate Smart', ports in the Pacific islands need to understand the relevant climate impacts and risks for their particular operation; only then can they determine what adaptation measures may be appropriate.

The Climate Smart Seaports web-tool is designed primarily for port personnel who make (or influence) decisions around long-term port planning for infrastructure, assets and management systems. However, it will also be of value to port owners and related businesses, government departments and local authorities concerned with ports and infrastructure; and for application by academic researchers.

The Climate Smart Seaports Tool enables interested users to begin the process of a climate risk assessment. It assists them to identify current and historical climate trends and variability, as well as future climate projections under a variety of scenarios.

Population and trade data is included, and users can add port-specific information to round out their analysis.

## Purpose of the document

The purpose of this document is to detail step by step how to create and integrate new datasets and data sources for the toolkit.

## Intended Audience and Reading Suggestions

This document is aimed to developers working – or willing to start working – on the project. It is assumed that a developer undertaking the creation of a new dataset for the toolkit is familiar with the project architecture and has the project set up in a development environment (see the document *Developer Documentation - Seaports Pacific*).

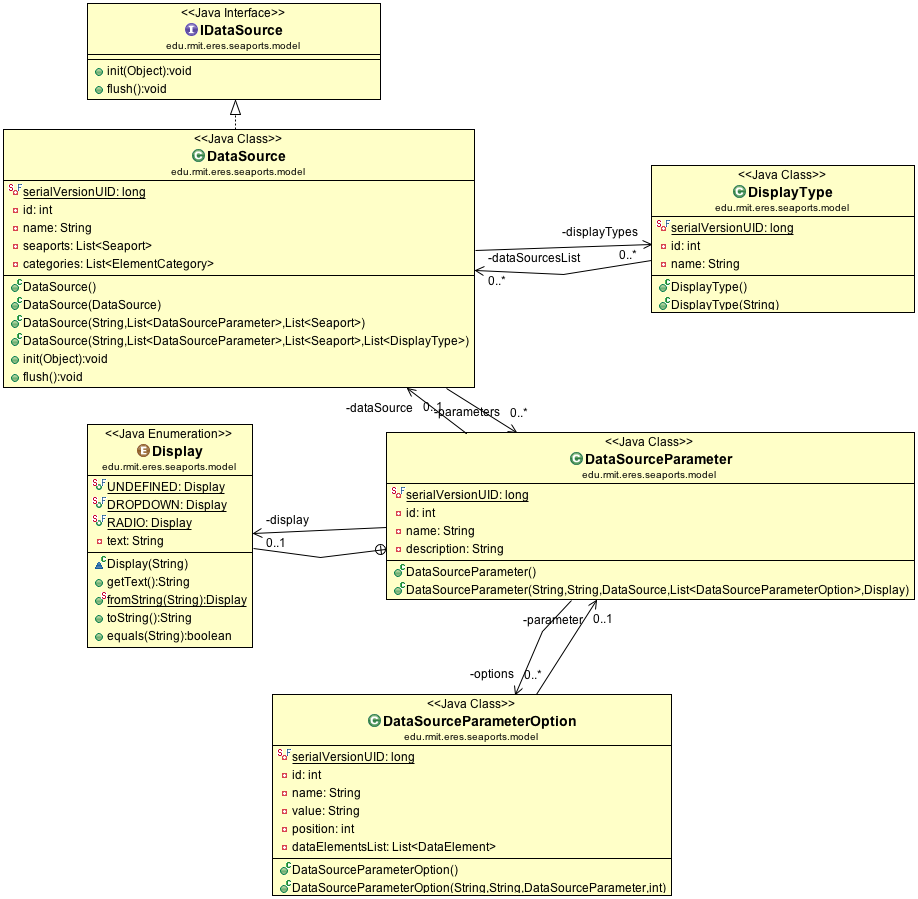
Since this is a step-by-step documentation, it is recommenced to read it following the section numbering.

## References

For documentation about installation of the development environment and deployment of the project, consult the *Developer Installation Guide* document.

# Overview

The design of the data sources for the Climate Smart Seaports project has been made more generic, and as much as possible decoupled from the system itself. Each data source is now inheriting from the *DataSource* class, implementing a common interface *IDataSource*.



Each data source is linked to one or more display types, defining which display types are available for this data source (for example, the “*Trade*” data source is linked to 2 display types: table and graph).

Data Sources have parameters, allowing the data source logic to retrieve the correct data from the data set(s) it accesses. Each of these parameters has a set of options available for the users to select when creating a data element.

The following table is the example of data in the database for the CSIRO Data Source. How to achieve this will be detailed step by step in this document.



Figure 1: Observed Trend data source and related database tables

A data element is referencing a data source and a set of parameter options selected by the user upon creation of the element.

The parameters can potentially be shared by multiple data sources, but most of the time, the parameters that apply to one data source are not applicable to another.

Based on the above data (Figure 1), the system generates the following form:

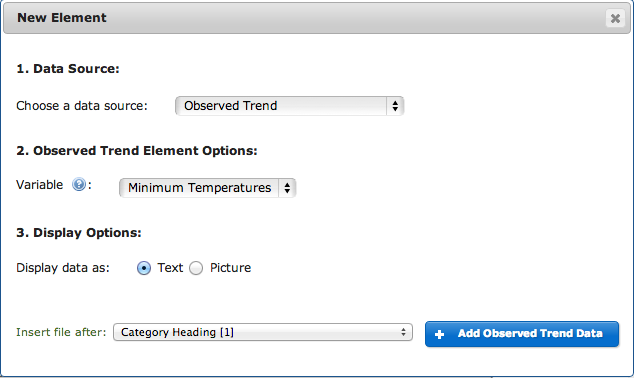


Figure 2: Observed Trend data source form generated based on data

When selecting the Observed Trend data source, the parameters are listed, with a help tooltip corresponding to the description field. The options available for each parameter are listed as defined by the “display” field. This allows complete customization of the forms based directly on the database content.

Finally, the display types available for the data source are listed. The selected one will define the way the data element will be displayed. The very last field “*Insert element after*” doesn’t depend on the data source but on the data elements existing in the report, in order to define the position of this new element. The options selected above (Figure 2) will produce a data element like this:

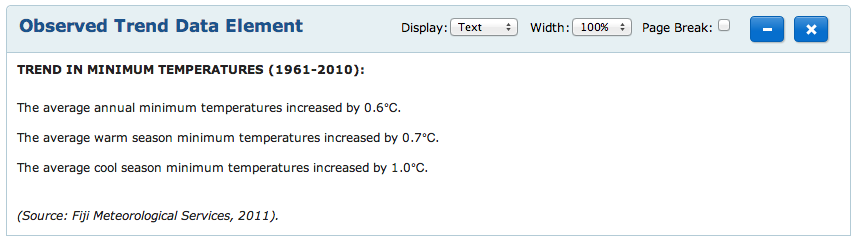


Figure 3: Data Element produced by the previous form

To create your own data source, you need to create a data source class and eventually the underlying data model, add the data source definition in the database, and create the views related to the data source.

# Create a new Data Source

## Create a Data Model (optional)

**This first step is optional:** **it is only needed if the dataset that your new data source is going to be stored within the Climate Smart Seaport database**. This is for example the case of the Observed Trend data source, for which data is contained in the database. Example of cases where this is not needed: if the data source accesses data within a file or through a web service, designing a data model is not necessary.

Create the Data Model(s) and the Data Access Object(s) to hold the dataset that your data source is going to access. The following diagram shows the data model for the Future Trend data:

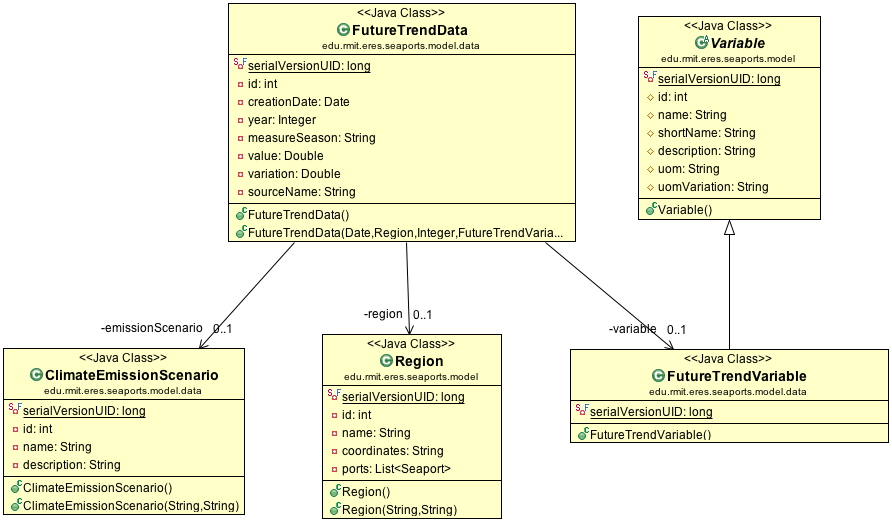


Figure 4: Example of the Data Model for the Future Trend Data Source

## Create a Data Source class

Create a new Data Source class, and make sure the name follows the convention followed in the project: “***TypeOfMySource***DataSource” (example: ***FutureTrend***DataSource). You can use the following template to create a new Data Source class:

Figure 5: template for a new Data Source class

@Entity

@DiscriminatorValue(value="**NameOfYourSource**")

**public** **class** **TypeOfTypeSource**DataSource **extends** DataSource **implements** Serializable {

/\*\*

\* Default constructor of data source

\*/

**public** **TypeOfMySource**DataSource() {

**super**();

}

/\*\*

\* Constructor of data source specifying all its fields but the display type

\* **@param** name: the name of the data source

\* **@param** parameters: the list of parameters for this data source

\* **@param** seaports: the list of seaports for which this data source is available

\*/

**public** **TypeOfMySource**DataSource(String name, List<DataSourceParameter> parameters, List<Seaport> seaports) {

**super**(name, parameters, seaports);

}

/\*\*

\* Copy constructor of data source

\*/

**public** **TypeOfMySource**DataSource(DataSource dataSource) {

**super**(dataSource);

}

/\*\*

\* Constructor of data source specifying all its fields

\* **@param** name: the name of the data source

\* **@param** parameters: the list of parameters for this data source

\* **@param** seaports: the list of seaports for which this data source is available

\* **@param** displayTypes: the display types available for this data source

\*/

**public** **TypeOfMySource**DataSource(String name, List<DataSourceParameter> parameters, List<Seaport> seaports, List<DisplayType> displayTypes) {

**super**(name, parameters, seaports, displayTypes);

}

@Override

**public** **void** init(Object obj) {

...

**My implementation of the ‘init’, called before getData()**

...

}

@Override

**public** **void** flush() {

...

**My implementation of the ‘flush’, called after getData()**

...

}

@Override

**public** List<**MyObjectClassToReturn**> getData(DataElement dataElement) {

...

**My Implementation to retrieve data based on the parameter**

...

**return** data;

}

}

The following snippet shows how the 3 methods (init, getData and flush) are used by the system:

**...**

// Instantiate the data source of the sub-type specified by the data source name

String className = "edu.rmit.eres.seaports.model." + WordUtils.*capitalize*(de.getDataSource().getName().toLowerCase()) + "DataSource";

Constructor<?> constructor = Class.*forName*(className).getDeclaredConstructor(DataSource.**class**);

constructor.setAccessible(**true**);

// Cast to the correct specific data source type

Class<? **extends** DataSource> dataSourceClass = Class.*forName*(className).asSubclass(DataSource.**class**);

DataSource ds = dataSourceClass.cast(constructor.newInstance(**new** Object[] { de.getDataSource() }));

// Retrieves the data and set the field

**if** (ds **instanceof** FutureTrendDataSource)

ds.init(futureTrendDataDao);

**try** {

List<?> data = ds.getData(de);

de.setData(data);

}

**catch** (NoResultException e) {

e.printStackTrace();

}

ds.flush();

**...**

Figure : edu.rmit.eres.seaports.controller.ReportController, method “Report prepareReportData(Report)”

As you can see from that snippet, the *init* method allows passing on a parameter to the data source.

In the case of *FutureTrendDataSource*, the object passed is a Data Access Object (DAO) Autowired by Spring MVC in order for the data source to have access of the Future Trend data from the database. This is only needed because the data is internally stored in the CSS database. In other cases, the *init* method might not even need to be called at all.

## Add Data Source definition in the database

Add your data source into the Java class responsible of the initial loading of the database. For this, you need to define:

* your data source
* the parameters for the data source and the option available for each parameter
* for which seaports it is available
* in which element categories it should appear

/\*\*

\* Loads the Future Trend Data Source in the database

\* **@param** session: the Hibernate Session object which takes care of persisting objects in the database

\*/

**public** **static** **void** LoadFutureTrendDataSource(Session session)

{

// Loads the underlying Observed Trend dataset

FutureTrendDataSourceLoader.*LoadFutureTrendData*(session);

// Display Types offered by this data source

DisplayType tableDisplayType = (DisplayType)(session.get(DisplayType.**class**, 3)); // Table

DisplayType pictureDisplayType = (DisplayType)(session.get(DisplayType.**class**, 5)); // Picture

List<DisplayType> displayTypes = **new** ArrayList<DisplayType>();

displayTypes.add(tableDisplayType);

displayTypes.add(pictureDisplayType);

// Data Source

ObservedTrendDataSource dsTrend = **new** ObservedTrendDataSource("futureTrend", "Future Trend", "", **null**, **null**, displayTypes);

// Parameter Climate Variable, with options Temperature, Rainfall and Relative Humidity

DataSourceParameter variableParam = **new** DataSourceParameter("Variable", "<p>?.</p>", dsTrend, **null**, DataSourceParameter.Display.*DROPDOWN*);

session.save(variableParam);

DataSourceParameterOption variableTemp = **new** DataSourceParameterOption("Temperature", "Surface Air Temperature", variableParam, 1);

session.save(variableTemp);

DataSourceParameterOption variableRainfall = **new** DataSourceParameterOption("Rainfall", "Total Rainfall", variableParam, 2);

session.save(variableRainfall);

// Availability of the data source for each seaport

List<Seaport> seaports = **new** ArrayList<Seaport>();

seaports.add((Seaport)(session.get(Seaport.**class**, "FJSUV")));

seaports.add((Seaport)(session.get(Seaport.**class**, "FJLTK")));

seaports.add((Seaport)(session.get(Seaport.**class**, "FJMAL")));

seaports.add((Seaport)(session.get(Seaport.**class**, "FJLEV")));

seaports.add((Seaport)(session.get(Seaport.**class**, "FJWAI")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGGUR")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGATP")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGBUA")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGDAU")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGKVG")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGKIE")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGKIM")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGLAE")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGLOR")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGMAG")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGROR")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGPOM")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGRAB")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGVAI")));

seaports.add((Seaport)(session.get(Seaport.**class**, "PGWWK")));

dsTrend.setSeaports(seaports);

// Availability of data sources for each element category

List<ElementCategory> categories = **new** ArrayList<ElementCategory>();

// Category 2 = Future climate & marine

categories.add((ElementCategory)(session.get(ElementCategory.**class**, 2)));

dsTrend.setCategories(categories);

session.save(dsTrend);

}

## Controller implementation (Optional)

This next step is optional and only applies to the data sources that need to access the CSS database using a DAO class to retrieve data (for example, the Future Trend data source).

If so, you will need to add the following code in the “*ReportController*” class:

At the top of the class definition, add an *Autowired* field to the DAO you need within your data source. Example for the Future Trend data source:

@Autowired

**private** FutureTrendDataDao futureTrendDataDao;

In the method “*Report prepareReportData(Report)*” of the *ReportController*, add a call to the init(…) method of your data source, passing on the DAO object you have declared as *Autowired* at the top of the class. Example for the Future Trend data source:

if (ds instanceof FutureTrendDataSource)

ds.init(futureTrendDataDao);

This way, when the controller will call the *getData(DataElement)* method of your data source, the DAO object will be used to retrieve the data from the database.

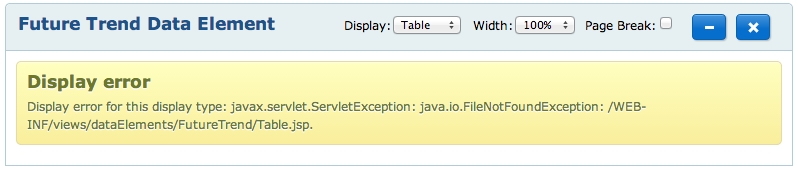
## Create Views for the display types

Now that the data source is all ready, we need to display it in the tool. For this, we have defined which display types will be associated to the data source in section 3.3. For example, the Future Trend data source can be displayed as a table or as a picture.

For each of the display types available, the JSP file responsible of displaying data elements, (*elements.jsp*) calls a specific JSP file for displaying the data source, based on the data source’s type and the selected display type’s name:

<jsp:include page=*"dataElements/*${formattedDataSourceName}*/*${element.displayType}*.jsp"* />

If the JSP view doesn’t exist, a warning message like the one below is displayed:



For example, if your data source type is “**MySourceType**” and you have defined the display types *Table* and *Pictures*, you need to define the following JSP view files:

*src/main/wepapp/WEB-INF/views/dataElements/***MySourceType***/***Table.jsp**

*src/main/wepapp/WEB-INF/views/dataElements/***MySourceType***/***Picture.jsp**

In these files, you can implement how to display the specific data of your data source for each display type.