Tropical Water Quality Hub

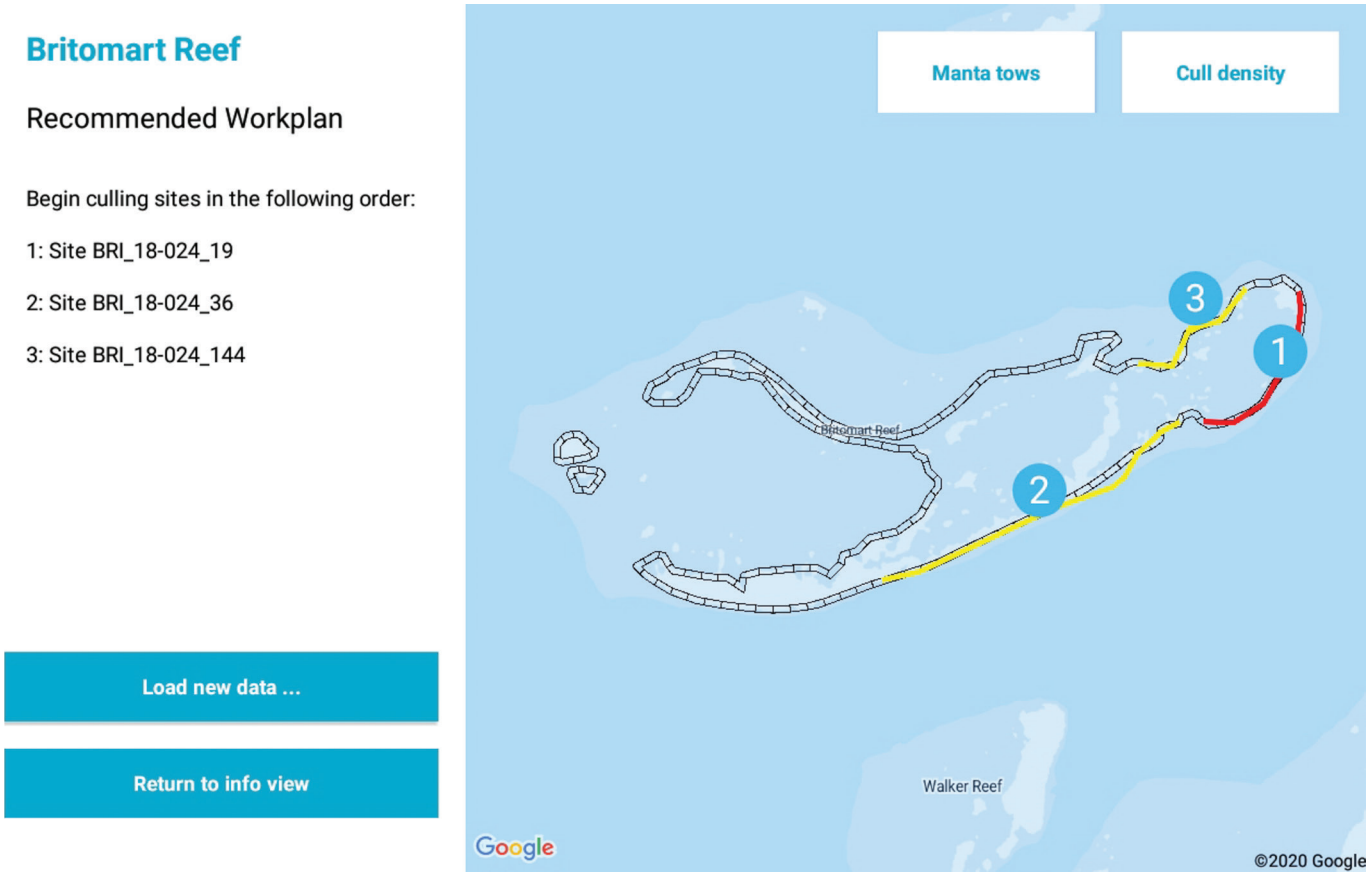
National Environmental Science Programme

Technical Report

# The COTS Control Centre

## Supporting ecologically-informed decision making when and where decisions need to be made

Cameron S. Fletcher and David A. Westcott




CSIRO

# The COTS Control Centre

## Supporting ecologically-informed decision making when and where decisions need to be made

Cameron S. Fletcher and David A. Westcott

CSIRO Land & Water



Supported by the Australian Government's
National Environmental Science Program
Project 5.1: Matching the Crown-of-Thorns Starfish Integrated Pest Management to the scale of the new Control Program

National Library of Australia Cataloguing-in-Publication entry:
978-1-925514-80-3

This report should be cited as:
Fletcher C. S. and Westcott D. A. (2021) *The COTS Control Centre: Supporting ecologically-informed decision making when and where the decisions need to be made*. Report to the National Environmental Science Program. Reef and Rainforest Research Centre Limited, Cairns (189pp.).

Published by the Reef and Rainforest Research Centre on behalf of the Australian Government's National Environmental Science Program (NESP) Tropical Water Quality (TWQ) Hub.

The Tropical Water Quality Hub is part of the Australian Government's National Environmental Science Program and is administered by the Reef and Rainforest Research Centre Limited (RRRC). The NESP TWQ Hub addresses water quality and coastal management in the World Heritage listed Great Barrier Reef, its catchments and other tropical waters, through the generation and transfer of world-class research and shared knowledge.



Cover photographs: (front) The COTS Control Centre. (back) COTS control divers. Images: Cameron Fletcher.

This report is available for download from the NESP Tropical Water Quality Hub website:
http://www.nesptropical.edu.au

# CONTENTS

Contents....i
List of Figures....iv
Glossary....v
Acknowledgements....vi
Executive Summary....1
1.0 Introduction....2
2.0 Design philisophy....3
2.1 Overarching architecture....3
2.2. Database design....3
2.2.1 Database Tables....3
2.2.1 Database Table Relationships....4
2.3 App design....5
2.3.1 User Interface....5
2.3.2 ContentProvider and Loaders....9
2.3.3 Custom types....10
2.3.4 Decision tree class....12
2.4 Interaction between apps....12
2.4.1 File system structure....13
2.4.2 Importing data....13
2.4.3 Processing imported data....13
2.4.4 Linking imported data to existing data....14
2.4.5 Geospatial analysis....15
2.4.6 Record checking....16
2.4.7 Adding new records to the database....16
2.4.8 Updating the User Interface....17
3.0 The decision support tool....18
3.1 Overview of an ecologically-informed underpinning for control....18
3.3 Implementing the Decision Tree....21
3.3.1 Associating manta tows with Sites....22
3.3.2 Assigning Reefs to Intensive or Maintenance mode....23
3.3.3 Informing the decision to manta tow....23
3.3.4 Informing the decision to cull....24
3.3.5 Informing the order in which Sites are culled....25
4.0 Summary....26

References....27
Appendix A: ThinkSpatial json structures....29
A.1 culldata.db – culldata Table – json field....29
A.2 surveillance.db – surveillance Table – json field....30
A.3 rhisdata.db – rhis Table – json field....32
Appendix B: Source Code....39
B.1 MainActivity.java....39
B.2 DisplayMap.java....53
B.3 DisplayInfo.java....66
B.4 LoadNewData.java....73
B.5 ImplementDecisionTreeAtReef.java....84
B.6 FindMantaNearestSite.java....93
B.7 COTSDataContract.java....98
B.8 COTSDataProvider.java....102
B.9 COTSDataHelper.java....116
B.10 Dive.java....120
B.11 DiveList.java....127
B.12 Manta.java....134
B.13 MantaList.java....140
B.14 Site.java....149
B.15 SiteList.java....151
B.16 SitePolygon.java....155
B.17 SitePolygonList.java....157
B.18 SitePolygonPoint.java....160
B.19 SitePolygonPointList.java....163
B.20 Reef.java....169
B.21 ReefPolygon.java....172
B.22 ReefPolygonPoint.java....173
B.23 Voyage.java....175
B.24 Vessel.java....177
B.25 Rhis.java....179
B.26 main_activity.xml....181
B.27 display_map.xml....182
B.28 display_info.xml....183
B.29 site_marker_info.xml....185
B.30 styles.xml....185

B.31 colors.xml .....188
B.32 strings.xml .....188
B.33 circle.xml .....189

# LIST OF FIGURES

Figure 1: Database Structure ..... 4
Figure 2: The opening view of the CCC-DST app ..... 6
Figure 3: User Interface workflow of selecting Reef for control ..... 6
Figure 4: ReefInfo view ..... 7
Figure 5: Exploring data to focus on most recent a) cull Dives, or b) Manta tows ..... 7
Figure 6: Generate Workplan ..... 8
Figure 7: Load new data ..... 9
Figure 8: The simplified on-water decision tree ..... 20

# GLOSSARY

**AMPTO** ..... Association of Marine Park Tourism Operators
**Control Program** ..... A coordinated COTS Control Program run on the GBR
**COTS** ..... Crown-of-thorns starfish (*Acanthaster* cf. *solaris*)
**COTS Control Centre** ..... Advanced tablet-based DSS being developed for the NESP COTS IPM Research Program, consisting of the hardware, three data collection apps built for GBRMPA by ThinkSpatial, and three app components developed by CSIRO for NESP
**CCC-DSS** ..... See COTS Control Centre
**CCC-DST** ..... The Decision Support Tool component of the CCC-DSS developed by CSIRO for NESP
**CCC-DE** ..... The Data Explorer component of the CCC-DSS developed by CSIRO for NESP. This functionality is currently built into the CCC-DST but may, in future, be split into a standalone app.
**CCC-DS** ..... The Data Sync component of the CCC-DSS develop by CSIRO for NESP
**CPUE** ..... Catch-per-unit-effort, in COTS per minute bottom time
**Dive** ..... A single control dive at a Site on a Reef, typically of 40 minutes bottom time duration
**DSS** ..... Decision Support System
**Ecological Threshold** ..... The COTS density below which coral growth can outpace COTS damage at a Site, in this report typically expressed in units of CPUE
**Expanded Control Program** ... The new Expanded Control Program run since November 2018
**GBR** ..... Great Barrier Reef
**GBRMPA** ..... Great Barrier Reef Marine Park Authority
**Historical Control Program** .... The Control Program delivered between 2012 – 2018
**Intensive Control** ..... Reef above the Ecological Threshold, revisited frequently for cull actions
**IPM** ..... Integrated Pest Management
**Maintenance Mode** ..... Reef below the Ecological Threshold, revisited periodically for manta tow
**NESP** ..... National Environmental Science Program
**Priority Reef** ..... A Reef selected for control actions due to its high ecological and/or economic value
**Reef** ..... A single reef on the GBR, typically with its own unique identification code
**RRRC** ..... Reef and Rainforest Research Centre
**Site** ..... A single site on a single Reef in the GBR, typically 10ha in size
**Vessel** ..... Boat of contractor employed to undertake COTS control
**Voyage** ..... A single Control Program voyage, typically lasting 10 days
**Voyage Plan** ..... List of Reefs, in order, to be visited on the current Voyage

# ACKNOWLEDGEMENTS

The Association of Marine Park Tourism Operators (AMPTO), the Reef and Rainforest Research Centre (RRRC), and the Great Barrier Reef Marine Park Authority (GBRMPA) provided access to data and important contextual information about the field operations of the control program. In particular, Steve Moon and Col McKenzie, AMPTO, were always keen to critique our work and in doing so made valuable contributions. This project is supported with funding from the Australian Government's National Environmental Science Program (NESP) Tropical Water Quality (TWQ) Hub.

# EXECUTIVE SUMMARY

This report outlines the design principles and implementation of the COTS Control Centre Decision Support Tool (CCC-DST) within the COTS Control Centre Decision Support System (CCC-DSS). The CCC-DSS is a combined hardware and software solution developed by CSIRO as part of the National Environmental Science Program (NESP) Integrated Pest Management (IPM) Crown-of-thorns starfish (COTS) Research Program to help guide on-water decision making and implement the ecologically-informed management program outlined in the report *An ecologically-based operational strategy for COTS Control: Integrated decision making from the site to the regional scale* (Fletcher, Bonin, & Westcott, 2020).

The COTS Control Centre DSS is built around a fleet of 32 ruggedised Samsung Galaxy Tab Active2 Android tablets, along with a suite of three data collection apps, developed for the Great Barrier Reef Marine Park Authority (GBRMPA) by ThinkSpatial, and three decision support components developed by CSIRO as part of the NESP COTS IPM Research Program. The fleet of tablets are able to be managed remotely, including locating hardware and updating software, using the Samsung Knox Manage Enterprise Mobility Management platform, and run a custom kiosk launcher. Data is shared between the apps that make up the CCC-DSS within a tablet using the Android file system, between tablets on a vessel independent of cellular connectivity with the Android Nearby Communications protocol, and with GBRMPA's Eye on the Reef Database when cellular networking is available.

In addition to designing and implementing the overarching CCC-DSS system, CSIRO has developed a suite of three software components, consisting of the main Decision Support Tool (CCC-DST), a Data Explorer functionality, which is currently implemented as part of the CCC-DST but may, in future, be separated into a second app, and a utility Data Sync Tool for sharing data between tablets when internet connectivity is not available. This report details the underlying philosophy and implementation notes of the CCC-DST. It outlines the methods by which the principles outlined in *An ecologically-based operational strategy for COTS Control* (Fletcher, et al., 2020) were encoded into the CCC-DST application. Full source code for the app is provided in the Appendices.

# 1.0 INTRODUCTION

Crown-of-thorns starfish (COTS) outbreaks are one of the major threats to coral in the Great Barrier Reef (GBR) and across the Indo-Pacific (Bruno & Selig, 2007; De'ath, Fabricius, Sweatman, & Puotinen, 2012; Pearson, 1981; Pratchett, Caballes, Rivera-Posada, & Sweatman, 2014; Yamaguchi, 1986). However, unlike other major threats facing coral reefs, such as bleaching (Claar, Szostek, McDevitt-Irwin, Schanze, & Baum, 2018; Hoegh-Guldberg, 1999; Hughes, 2003; Hughes et al., 2018a; Hughes, Kerry, & Simpson, 2018b) or cyclone damage (Gardner, Côté, Gill, Grant, & Watkinson, 2003; Pratchett et al., 2014), COTS populations and their impacts can be immediately reduced in specific locations using localised control methods (Fletcher et al., 2020; Westcott & Fletcher, 2018).

The primary method of COTS control currently available is manual culling of individual starfish by divers through single-shot injection with bile salts or vinegar solution (Firth & McKenzie, 2015; Moutardier et al., 2015; Rivera-Posada, Pratchett, Cano-Gómez, Arango-Gómez, & Owens, 2011). When applied at appropriate spatial and temporal scales, this approach has been shown to generate meaningful reductions in COTS populations and corresponding recovery or maintenance of coral cover at sites within reefs (Westcott & Fletcher, 2018), and, more recently, over entire reefs (Westcott et al., 2020). However, the fact that it depends on divers injecting individual starfish makes control manually intensive and expensive to apply over large areas. This makes it vitally important that control investment is focussed at high priority locations and at an intensity where it can achieve ecologically-meaningful outcomes.

Early work in the NESP COTS Integrated Pest Management (IPM) Research Program demonstrated that data collected by the Control Program could be used to improve efficiency by targeting control actions within a reef (Fletcher & Westcott, 2016). Building on this, in 2018 CSIRO developed, with input from reef managers at GBRMPA and on-water control staff, a full ecologically-informed framework for the expanded National COTS Control Program. The design of this framework was described in detail in Fletcher et al. (2020). That report defined decision trees to ensure that all the day-to-day on-water decisions required to run the Control Program could be made in an ecologically-informed manner, leveraging data collected by the Control Program itself. The first iteration of the framework was designed to be implemented manually by decision makers on-water through consideration of the data they collected, however the report also recommended the development of an on-water Decision Support System to automate Control Program data collection and analysis, and lay the groundwork for more advanced decision support systems in future.

This report details the design and implementation of the COTS Control Centre Decision Support Tool (CCC-DST) designed to facilitate this vision within the COTS Control Centre Decision Support System (CCC-DSS). The CCC-DSS is a hardware and software solution, comprising 32 Samsung Galaxy Tab Active2 tablets, managed using Samsung Knox Manage Enterprise Mobility Management System. The tablets run a kiosk operating system and provide three data collection apps, developed for GBRMPA by ThinkSpatial, and three decision support components developed by CSIRO as part of the NESP COTS IPM Research Program. This report lays out the design principles, describes how the ecologically-informed principles underpinning the National COTS Control Program were implemented in the CCC-DST, and includes the source code of the software developed as Appendices.

# 2.0 DESIGN PHILISOPHY

## 2.1 Overarching architecture

In general, Android applications are designed to require minimal memory footprint by cycling data out of their sqlite database on the fly as required. This is an efficient method of designing lightweight apps coexisting on mobile hardware will limited memory and processing capacity. It works well for some of the tasks required of the NESP COTS Control Centre apps, but is not intuitively suited to all the functionality required. This is especially the case for situations where data must be compared longitudinally through time or across many Sites or Reefs in order to guide decision making. On the other hand, because the COTS Control Centre is run on dedicated hardware of known specification and containing only the suite of apps necessary to guide on-water actions, we can bias the design of our system towards functionality within the hardware being used, rather than optimising for universal efficiency.

As a result, the COTS Control Centre apps were developed with a hybrid philosophy that aims to target the data loaded into memory to that required to inform a decision, and which structures the data in memory using custom classes that reflect the actual data being analysed. This approach puts an emphasis on both logical database design and well-structured custom types to support the functionality of the apps, each of which are described in further detail in the sections that follow.

The apps were developed in stages, starting with prototypes early in 2016. As a result, they retain some legacy functionality, such as the use of loaders rather than the ViewModel and Room functionality introduced in Android after this time. They also include components, such as a ContentProvider framework, that were expected to be important at one stage in app development, but which are not deeply leveraged in the current configuration. The structure of the apps also reflects the fact that their development will continue to incorporate feedback from on-water operators, as well as scientific advances in biological understanding, field measurements, and management strategies. As a result, in some places a more general programming approach has been favoured over more optimised code in order to maintain or provide flexibility to incorporate new functionality in future.

## 2.2. Database design

The overall database structure is illustrated in Figure 1, which shows the nine linked tables of the cotsData.sqlite database that underpins the app. Each Table row contains only single values (i.e. no array fields), and all links between Tables are arranged to be many-to-one.

### *2.2.1 Database Tables*

The data is stored in a sqlite database within the app. The database is structured based on the physical structure of the management system, and contains components related to geographic features and management actions. The Control Program operates at the intersection of these components, and so the way they are structured is important to the operation of the Decision Support Tool.

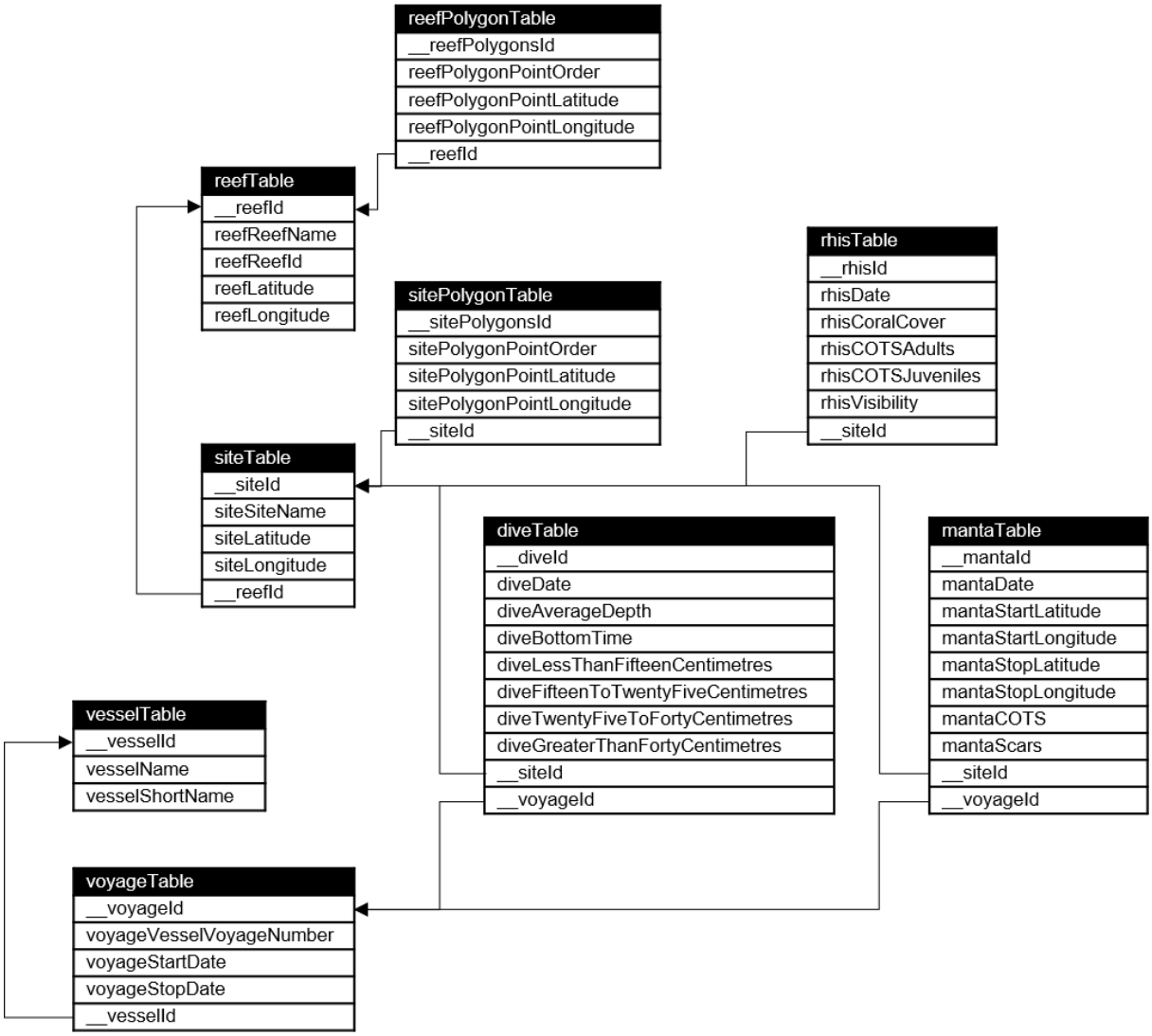



**Figure 1: Database Structure, illustrating the Tables contained in the main cotsData.sqlite database, the fields within those Tables, and the fields used to provide linkages between tables**

Geographic Features: The database contains Tables for storing two types of geographic features: Reefs and Sites. Although Sites are defined by the Control Program, once they are defined they are fixed. As such, the Site table, like the Reef table, should remain static and not contain any information related to management, other than the addition of new Sites.

Management Actions: The database contains Tables for storing five types of management-related information: Vessels, Voyages, Dives (cull dives), Mantas (manta tows), and RHIS (Reef Health Information Surveys). These characteristics reflect management actions: Dives report how many COTS were culled, Voyages report dates that they occurred etc. As management continues over time, more records will get progressively added to these Tables.

### *2.2.1 Database Table Relationships*

The inter-relationships between the various Tables are illustrated in Figure 1. Dependent relationships are indicated by an arrow pointing from the dependent Table field to the Table field in the Table on which it depends.

Reefs, Sites, and Vessels are physical objects that exist outside management actions. The ReefPolygonTable and SitePolygonTable contain the latitude and longitude points that define Reef and Site polygons, linked to the relevant record in the Reef and Site Table by reefId and siteId, respectively.

Voyages are linked to a specific Vessel by vesselId, and take place over a certain date period.

Dives take place on a Voyage and at a Site and take place on a certain Date. They record the number of COTS removed over a period of bottomTime over four size classes, <15cm, 15 – 25 cm, 25 – 40 cm, and > 40cm.

Mantas take place on a Voyage and at a "Nearest Site", and take place on a certain Date. Technically, Mantas take place at a Reef, and in future we may refine the database structure to reflect that because, ultimately, this is a derived relationship rather than a fundamental one, because Mantas are not strictly contained within Sites. Mantas record the number of COTS seen and a categorical record of how many COTS Scars are seen (a = “absent”; p = “present”, c = “common”).

RHIS take place on a Voyage and at a Site and take place on a certain Date. They are not used in the current version of the CCC-DST app, but the infrastructure is in place to allow the RHIS data to be explored, or leveraged to inform the decision tree process as it is refined in future.

## 2.3 App design

The app is designed as three core User Interface components, a set of loaders for loading data from various places in storage, a set of custom type classes, and a class for implementing the decision tree. These components are briefly outlined below.

### *2.3.1 User Interface*

The User Interface for the Decision Support Tool is kept intentionally simple, following the outcomes of testing more complicated prototypes. The goal of the DST is to present a single workflow, containing all the components required to make a decision and little unnecessary complicating information. However, the underlying structure of the application and user interface is designed to support both decision support and data exploration. Eventually, this data exploration functionality is likely to be split into its own application to maintain the simplicity of the CCC-DST application. In the current configuration, a data exploration functionality is provided within the CCC-DST app for viewing the most recent cull and manta results at a Reef.

The UI is made of three components, implemented as a MainActivity.java, and two fragments, DisplayMap.java and DisplayInfo.java. The source code for these classes may be referred to in the Appendices. The design philosophy is simple: MainActivity.java handles loading data and communication between the fragments; DisplayMap.java handles the display of the relevant loaded information on a Map, and handles user interactions, such as touches on the map or data exploration buttons; and DisplayInfo.java displays the relevant text information to the current stage of the workflow and presents buttons to the user to initiate the decision tree calculation. Compartmentalising the code like this facilitates debugging and reuse.

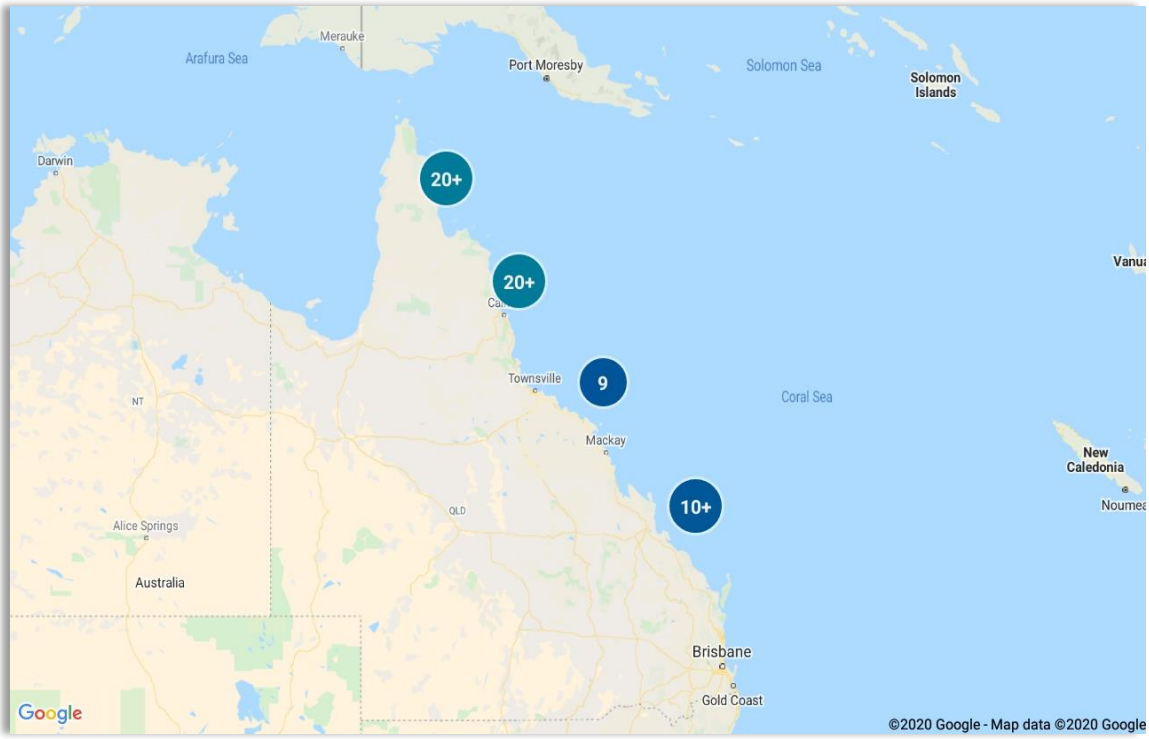



**Figure 2: The opening view of the CCC-DST app**

The basic workflow is guided by the new decision tree process, in which control decisions centre around Reefs. Upon opening the app, the user is presented with a view of all Reefs across the GBR for which control records currently exist (Figure 2). By default, all the Reefs for which any manta tow or control has previously occurred have pins, but in future the user will also be able to show all Reefs, including those that have never been visited previously, by selecting a “Show All Reefs” button in the top right corner of this view. The user can zoom and move around the map using familiar Google map gestures such as pinch-to-zoom and hold-and-drag, and as they zoom clusters resolve into individual pins (Figure 3). They can also hit the locator pin to have the app zoom to their current location using the GPS on the tablet. When the user has found the Reef they want to manage, they can click the pin to select a Reef at which to makes decisions. Up to this point, the DisplayInfo view and Load New Data and Generate Workplan buttons are not visible to the user, because no Reef has been selected.

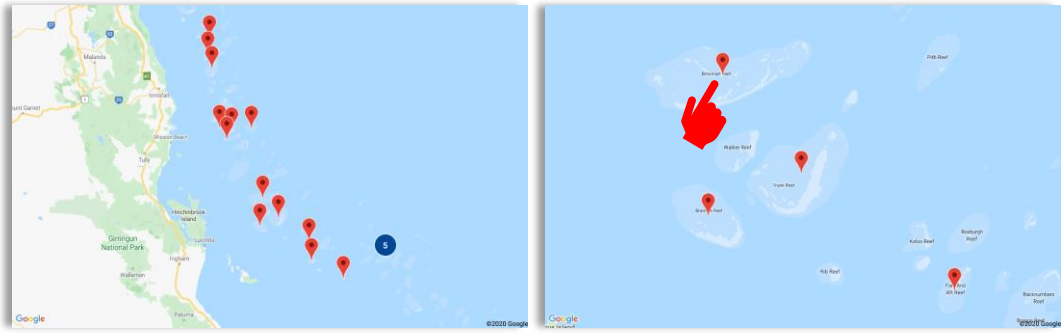


**Figure 3: User Interface workflow of selecting Reef for control. The user can zoom in from the opening view shown in Figure 2 using familiar Google map gestures such as pinch-to-zoom and hold-and-drag, and as they zoom clusters resolve into individual pins. They select a Reef to control at by clicked the Reef pin.**

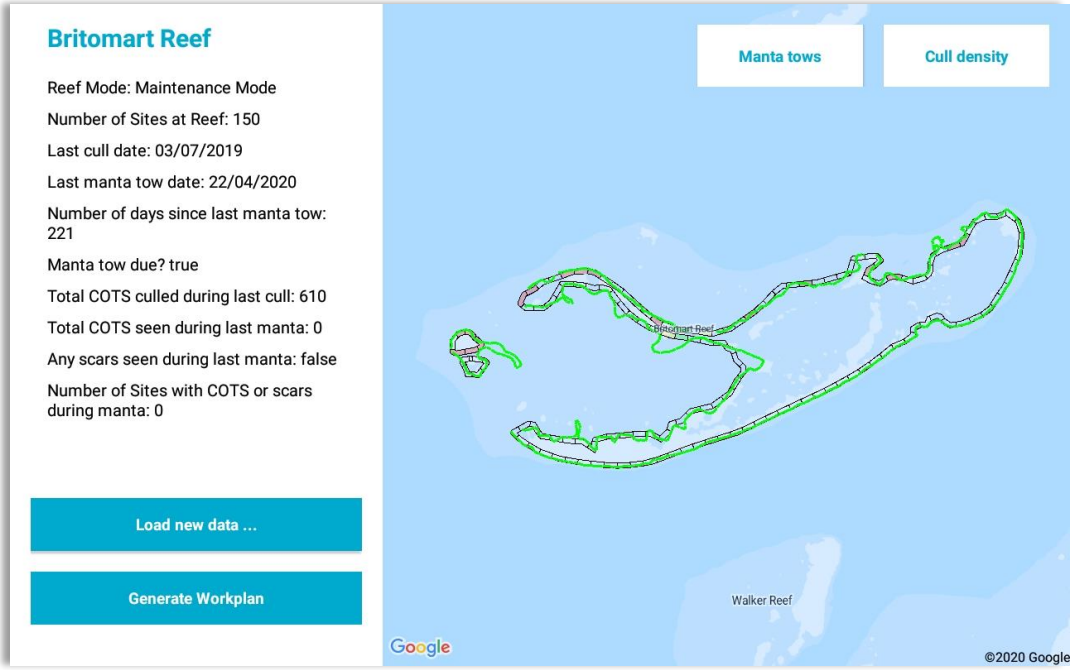



**Figure 4: ReefInfo view showing basic text information about the most recent Voyages to the Reef on which Dives or Mantas took place, plus a graphical view of the most recent Manta tows, and Sites coloured by the CPUE achieved there during the most recent Dive.**

Once the reef is selected 1) the DisplayInfo view, the Load New Data and Generate Workplan buttons become visible; 2) the map view zooms into the reef, and the Site polygons and the most recent CPUE and manta tow information is displayed in the DisplayMap panel; and 2) a summary of the most recent information collected at the reef is displayed in the DisplayInfo panel (Figure 4). The user can explore Reef data by turning on or off the display of most recent CPUE or manta data, to help them focus on specific information, using the “Manta tows” or “Cull data” buttons in the top right corner of the map view (Figure 5). In future, this will be refined to provide access to historical data and present metrics of performance over time as part of an expanded “Data Explorer” functionality.

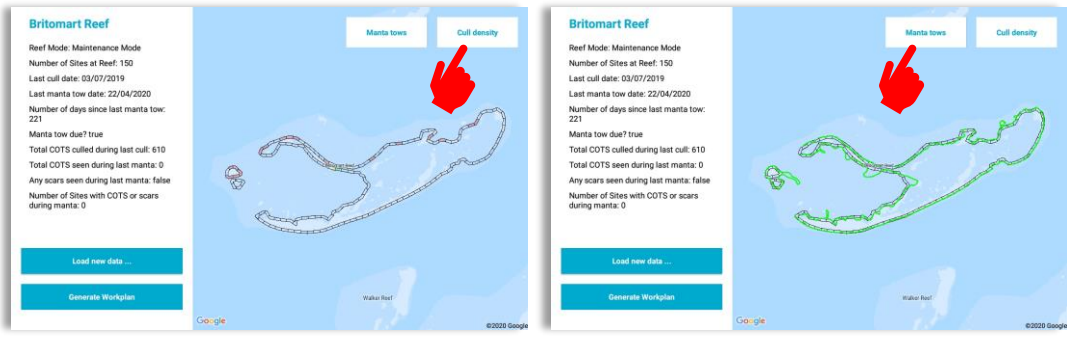



**Figure 5: Exploring data to focus on most recent a) cull Dives, or b) Manta tows**

The two decision support options available to the User at this point are “Load New Data” or “Generate Workplan”. Generate Workplan takes the data currently displayed, and runs it through the Decision Tree process described in detail in Fletcher *et al.* (2020), and outlined in Section 3.0 below, then presents the data: 1) as a text list of tasks, including whether the

current Reef needs to be Manta towed, and the Sites in the order that they should be culled; and 2) as a series of numbered pins on the map representing the order in which Sites should be culled (Figure 6).

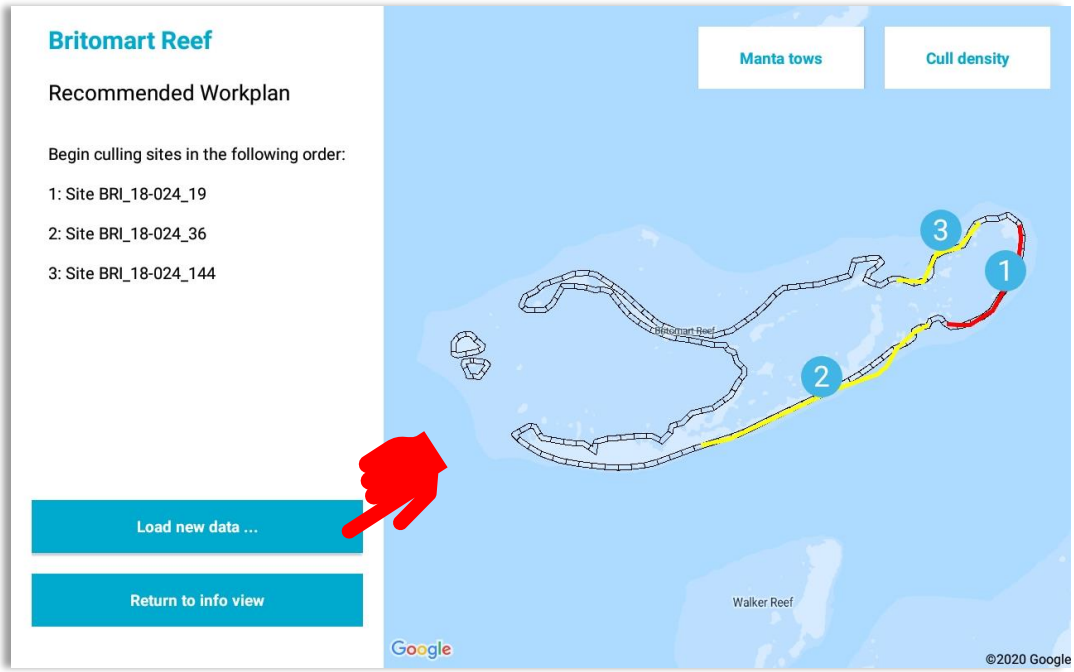



**Figure 6: Generate Workplan**

If new data has been collected in the partner data collections apps developed for GBRMPA by ThinkSpatial, then the user can select “Load New Data” to load it from the ThinkSpatial app into the CCC-DST. In this way, the latest manta tow or cull data can be loaded into the CCC-DST app and: 1) displayed and explored; and 2) used to update the Workplan for the Reef (Figure 7). An updated Workplan can be generated by selecting the Generate Workplan button after loading new data.

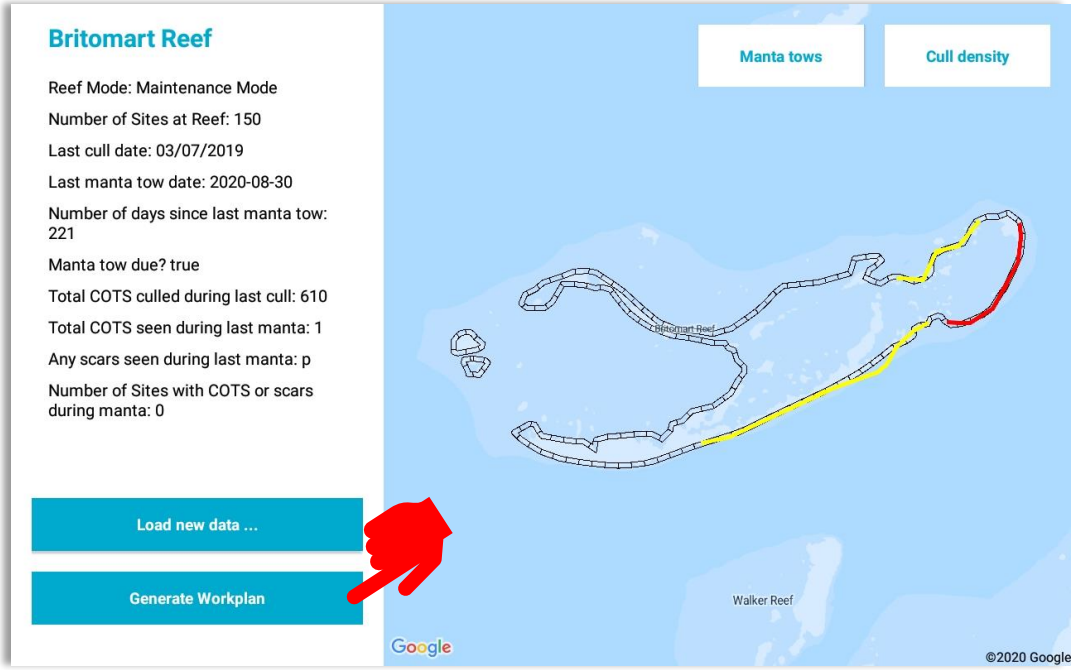



**Figure 7: Load new data**

These few steps: 1) selecting the current Reef; 2) generating a Workplan describing which task should be completed at the Reef next; 3) uploading the outcomes of manta and cull work just completed from the ThinkSpatial data collection apps; and 4) updating the Worklist in response to what was found is the key function for which the CCC-DST is designed, and the user interface reflects this workflow.

### *2.3.2 ContentProvider and Loaders*

There are two types of data that need to be loaded into the DST app: 1) app data stored in cotsData.sqlite in the native database structure described in Section 2.2; and 2) data exported from the ThinkSpatial apps stored in a simple database structure in which each record is stored as a JSON structure.

The CCC-DST was structured with a ContentProvider functionality early in its design when it was expected that the various apps that make up the COTS Control Centre Decision Support System (CCC-DSS) would require extensive interactive data sharing. In the current implementation, it is not highly leveraged, and in fact introduces additional complexities to the software design. However, the functionality has been maintained because it will be leveraged once the Data Exploration component of the CCC-DST is split off into its own app, and, eventually, if the ThinkSpatial apps need updates from the new GBRMPA Eye on the Reef database. However, we don't go deeply into the theory of its implementation as it relates to sharing data outside the CCC-DST app in this report.

Data from the main app database, cotsData.sqlite is loaded from the app's protected storage area when the app first starts. A variety of Loaders are created for different basic queries, loosely related to the underlying database Table structure. In the current implementation, each loader is related to one database Table and one custom Type. This architecture was implemented while testing a parallelised loading approach. The app now uses serialised

targeted queries to load data responsively. In future versions of the app this structure may be rationalised.

Each loader has an associated SQL query defined in COTSDataProvider.java. Queries are defined using the variable names provided in the COTSDataContract.java file, rather than database Table or Field names directly to facilitate maintainability. Due to the overheads of loading and processing large datasets within the memory footprint available on Android, queries are structured, where possible, to select only the information required for the current task in the app. For instance, when the app is first started, all information for all Reefs is loaded, but not for Dives, Sites or Mantas. Once a Reef is selected by the user, all the Sites and SitePolygons at that Reef are loaded and, initially, the Dives and Mantas from the most recent Voyages on which they were each collected are loaded and their info displayed. In future, historical data for the Reef will be then loaded in the background so that it is ready for deeper exploration, or for more advanced decision tree analysis, without slowing down user interface response.

Although each Type is generally loading information from the relevant database Table, other pieces of information are required to select the subset of data required. For instance, Dives contain the siteId of the Site at which they occur, but no direct information about the Reef at which they occur. However, Sites contain information about the reefId at which they occur. In order to facilitate the selection of Dives at a Reef, therefore, the queries make extensive use of SQL JOINS to tie the information required from various Tables together. In the previous example, the DiveTable would be JOIN to the SiteTable on siteId, and then Dives at Sites with the appropriate reefId selected for loading. In a similar way, the DiveTable may be simultaneously JOIN with the VoyageTable on voyageId, and SELECT results may be GROUP BY voyageId, ORDER BY voyageTable.startDate and then then LIMIT to the first 6 results in order to avoid having to load all the Dives that have ever occurred at the Reef. The specific implementations can be investigated in full in the source code appendices.

The results of the query are returned as a Cursor object. Each custom Type has a constructor that accepts a Cursor, as described below, and it references the COTSDataContract to establish the relevant column names for the associated Table of the database. Following the query, the cursor is iterated, a new custom Type object is created for each argument returned by the cursor, and added to the relevant custom TypeList, also as described below.

### *2.3.3 Custom types*

Much of the functionality of the app is provided by the custom Types defined for each type of data dealt with in the Control Program, and the custom methods provided within these Types. There are seven base types, based on the database structures outlined above: 1) Reef; 2) Site; 3) Vessel; 4) Voyage; 5) Dive; 6) Manta; 7) RHIS. In addition, there is a custom TypeList type for each of these components, which are primarily designed to speed up certain access operations and house custom methods that one could reasonably expect to want to be able to analyse from lists of the various custom Types.

The custom Types generally consist of:

1. Internal private variables matching the fields of the database Table from which they are generated.
2. At a minimum three constructors:
    a. a) empty;
    b. b) by passing the value of each required internal variable; and
    c. c) by Cursor
3. Getter functions for each internal variable
4. Setter functions for only those internal variables that there may be a need to modify after creation
5. Custom methods for accessing composite results physically meaningful for the customType; for instance the .totalCOTS() removed or .cpue() achieved on a Dive
6. Custom comparators for different types of comparison, such as comparing Dives by diveDate or by totalCOTS

The custom TypeList types provide functionality for lists of the custom Types beyond that available from simple Lists. The most important is fast lookup of each Type by its typeId, i.e. looking up a Dive by its diveId, or a Site by its siteId. In the database tables, the typeId is also the Primary Key, and so a Site can immediately be looked up by using its siteId as the index in a get method. This would work for the CCC-DST app too if every SiteList always contained all Sites in siteId order. However: 1) it would be infeasibly slow and impractical to load all Site and other data at all times; and 2) there are many reasons to make SiteLists to store intermediate results, such as the Sites at a single Reef or the Sites visited on a single Voyage. As soon as a list of Sites contains only a subset of all Sites, getting a Site by its siteId becomes a slow process requiring iteration through every item in the list and comparison of its siteId value with the target until the desired Site is located. Instead, our custom TypeList classes define a HashMap lookup table of typeId against index so that the relevant index can be immediately found for any typeId and the relevant object returned. The HashMap is updated when the TypeList is created and any time it is updated. For short lists this is possibly inefficient, future updates may optimise the implementation of these methods, however those updates should remain transparent to the rest of the application.

As noted above, the other goal of custom TypeLists is to house custom methods that one could reasonably expect to want to be able to analyse from lists of the various custom Types. For instance, accessing all the Sites within a SiteList from a specific Voyage, or being able to find the most recent Manta in a MantaList, or the Dive which culled the greatest number of COTS in a DiveList. The use of these custom TypeList classes significantly simplifies code throughout the rest of the app.

### *2.3.4 Decision tree class*

This class where the decision tree is implemented, ImplementDecisionTreesAtReef.java, takes as arguments when it is constructed:

1. the name of the Reef;
2. the siteList of all Sites at the Reef;
3. the diveList of all Dives at the Reef since the last Manta tow
4. the most recent Mantas at the Reef

Several methods return different information from the decision tree, including:

1. A text listing of the next control actions to take, including:
    a. Manta tow Maintenance Reef to ensure it remains below the Ecological Threshold;
    b. Manta tow Intensive Control Reef, before further recommendations on Site cull order are provided
    c. Recommendations on which Sites require culling, and in which order, given the most recent Manta and Dive data available within the CCC-DST
2. A numerical listing of the siteIds of Sites to be culled, and in which order; this information is used to update the map display

The exact selection of records at each step in the implantation of the decision tree, and the order in which they are processed, is vitally important, and the reader is referred to the fully commented source code in the relevant Appendix to understand the implementation details. The underlying logic and ecological-underpinning of the decision tree is described in detail in Fletcher et al. (2020), and the details relevant to its implementation in the CCC-DST app are described in detail in Section 3.0 below.

## 2.4 Interaction between apps

In order for the CCC-DST to incorporate new data from manta tows or culls to update its recommendations, it must import records from the data collection apps developed for GBRMPA by ThinkSpatial.

The ThinkSpatial apps can export records into .xls and .db files, either manually via the user interface or automatically as new records are produced. The CCC-DST imports data from the .db format files. The ThinkSpatial apps also upload data to GBRMPA's Eye On the Reef database when they can establish an internet connection. To facilitate this, they implement a buffered export strategy that stores records for upload until an internet connection can be established. In the export files, records are separated into two tables based on whether they have been submitted to Eye on the Reef at the time of export; a "data" Table and a "data_submitted" table.

Both tables store each record as a simple JSON text string, consisting of nested <key, value> pairs. The structure of these JSON strings is provided as an Appendix to this report.

In order to import the data from the ThinkSpatial apps, the CCC-DST:

1. Identifies the external directory in which the ThinkSpatial app exports are stored
2. Reads in the relevant .db file
3. Reads in each Table in the file
4. For each record, extracts the JSON string
5. Processes the JSON string to create the relevant CCC-DST internal type
6. Checks to see whether the record currently exists in the CCC-DST mysql database
7. If not, adds the record to the CCC-DST mysql database
8. Updates the map and info displays
9. Allows the user to generate an updated Worklist using the new data.

Some of these steps are relatively easy, most involve error checking and handling, and some are relatively complicated. We briefly run through the most important details below.

### *2.4.1 File system structure*

Android does not make accessing data exported by other applications easy. On Samsung devices like the Samsung Galaxy Tab Active2 tablets used as the hardware component of the COTS Control Centre Decision Support System, the relevant export directory is stored on the external SD card, in a directory with named with eight apparently random hex characters in two clusters of four separated by a dash. The directory name appears to be unique for each tablet. In order to identify the name of this directory on each tablet on which the CCC-DST is installed, we search the external SD card directory for directories matching the regular expression "(.*\\p{XDigit}{4}-\\p{XDigit}{4})".

Within this directory, the ThinkSpatial apps use the same child directory structures across all tablets:

1. ..\au.gov.gbrmpa.cots.capture\files\culldata.db for cull Dive data
2. ..\au.gov.gbrmpa.cots.surveillance\files\surveillance.db for Manta data
3. ..\au.gov.gbrmpa.cots.rhis\files\rhisdata.db for RHIS data

### *2.4.2 Importing data*

Data from each of the ThinkSpatial export files is read in using a standard Android SQLiteDatabase rawQuery and processing the data from the returned Cursor. This is completed once for each Table in each export file. Because the structure of the exported database is so simple, including only an id and json field for each record, reading the data from the database is simple, and identical across the three ThinkSpatial data collection apps. All bespoke processing occurs instead in accessing and interpreting the JSON string.

### *2.4.3 Processing imported data*

The bulk of the work importing the ThinkSpatial data takes place in processing the JSON string comprising each record. This outline of this process is the same across the three data collection apps, but the details differ. We use the org.json Android library to mediate access to the JSON String as JSONObjects.

JSON is structured as a series of nested <key, value> pairs; that is, a value may consist of an array of other <key, value> pairs. The entire JSON structure is stored as a String. Keys are strings, and values can be interpreted as a number of different types, including arrays of numbers or a list of nested JSON <key,value> pairs, as the JSON string is processed. In this way, data of almost arbitrary complexity can be stored hierarchically within the JSON string.

Processing the JSON involves:

1. For the case of simple <key, value> pairs; reading the appropriate value using the hardcoded “key” and casting to the appropriate type using the appropriate getter method
2. For the case of composite <key, value> pairs where the value is a list of nested JSON <key, value> pairs; reading the appropriate nested value list using the hardcoded “key”, casting to JSONObject, and then within that JSONObject, reading the appropriate value using the hardcoded “key” and casting to the appropriate type using the appropriate getter method. Some nested hierarchies are several levels deep and this process must be repeated for each nested layer.
3. For the case of composite <key, value> pairs where the value is an array; reading the appropriate nested array using the hardcoded “key”, and casting to JSONArray, then accessing the relevant entries of the array using traditional get methods or iterators.

### *2.4.4 Linking imported data to existing data*

Although not conceptually related to processing JSON, this part of the code is also where we cross-reference the new data with the old data. For instance, each Dive that is imported must be matched to the relevant Site record in the CCC-DST sqlite database. This is done using a SQLiteDatabase rawQuery on the siteName field of the siteTable of the cotsData.sqlite database against the cullZone -> “sitename” field from the ThinkSpatial JSON file, returning the siteId of the site in the cotsData.sqlite siteTable.

This process can fail for several reasons. One of these arises when the data relates to a new Site created in the ThinkSpatial apps but not yet incorporated in GBRMPA’s Site Polygon records or the CCC-DST apps. At the moment, these records are omitted from import as an edge case, but in future the Site Polygon will be added temporarily to the CCC-DST cotsData.sqlite sitePolygonTable from the cullSite -> geometry data incorporated in the ThinkSpatial export file, before being permanently added to the database during the next GBRMPA Eye on the Reef data refresh.

A second issue arises when the ThinkSpatial app does not assign the manta tow to any cullSite. We do not know what internal code ThinkSpatial uses to assign manta tows to cullSites, but presumably if the manta tow deviates beyond a certain distance from any cullSite it is simply not assigned to any cullSite. As we use our own robust code to assign manta tows to Sites, this would not be an issue, except that ThinkSpatial does not export detail of the Reef at which the manta tow occurred except in the cullSites data; so if there are no assigned cullSites there is no information about the Reef at which the manta tow occurred. In the immediate term, this is treated as an edge case and the data dropped. In the short term, a workaround will be implemented to assign the manta to the nearest Reef based on the latitudes and longitudes of its towline. In the medium term, ThinkSpatial will need to update their app to

export information about the Reef at which the manta tow was conducted, even if it was not close enough to be assigned to any Site at that Reef.

A similar process is also used to assign Dives and Mantas to their respective Voyages in the cotsData.sqlite voyageTable.

### *2.4.5 Geospatial analysis*

Once the manta tows are imported, they must be assigned to their nearest Site in order to be used by the Decision Support Tool. The ThinkSpatial data collection apps appear to assign manta tows to cullSites, but the mechanism is unknown and manta tows often seem to be assigned across multiple Sites, even though they should usually only intersect one or at most two Sites, given that their average length is around 200 m, compared to an average Site length of around 500 m. Regardless, we use our own algorithm to assign manta tows to Sites.

To do this, we need to georeference the manta towline, based on its latitude and longitude, against the latitude and longitude of the polygons defining each Site at the Reef at which the tow occurred, before calculating which Site Polygon the Manta tow is closest to. The computation is complicated by the fact that manta tows do not have a simple spatial relationship with Sites: they are not contained within Sites, they can intersect multiple Sites, or they can intersect no Sites but be “nearby” one or several Sites. Georeferencing spatial data is a common process available in desktop Geographic Information Systems, but we need a method capable of operating on-water on the tablet hardware on which the COTS Control Centre is built.

To achieve this, we leverage a hybrid approach in which the computationally expensive part of the process is pre-calculated ahead of time, so that manta tows can be quickly and accurately assigned to cull Sites as data from the ThinkSpatial surveillance app is imported into the CCC-DST. The details are outlined in Section 3.3.1 below, but in short, lookup raster tables were precalculated for the 313 Reefs at which GBRMPA has defined Sites, to provide immediate estimate of the nearest Site for any latitude and longitude within a boundary area of each reef.

In terms of implementation, when new surveillance data is imported, for each manta tow:

1. The Reef of the manta tow is extracted from the ThinkSpatial JSON data field
2. The lookup table for that Reef is loaded into the app memory
3. The start and stop latitude and longitude of the manta towline are used to estimate the mean latitude and longitude
4. That mean latitude and longitude is converted to a raster coordinate using the metadata header information in the lookup table
5. The nearest Site to the manta tow is read from the lookup table at the calculated raster coordinate

This system is fast, but assigns manta tows based on their median latitude and longitude, so doesn’t proportionally split manta tows between Sites when they cross Site boundaries. It is also precalculated, so cannot assign mantas to new Sites defined on-water using the ThinkSpatial apps. In future, the performance cost of implementing a true georeferencing capability will be explored further.

### 2.4.6 Record checking

In order to prevent the same records being added repeatedly to the main database of the CCC-DST application, it is necessary to test whether a record has already been added before adding it again. This is a non-trivial endeavour, because although unlikely, it is not impossible that two valid and distinct but, within the resolution provided by the structure of the databases involved, seemingly identical records could exist.

This is most likely to be an issue for Dives, in which case a Dive on the same day, at the same Site, could record the same cull numbers across two distinct Dives, especially if both Dives' cull counts are zeros across all size classes. Because the database only records Dive records to the nearest day, and because two Dives at a specific Site on a specific Voyage will have common __siteId and __voyageId, and are likely to have common averageDepth and diveBottomTime, Dives with the same cull numbers at the same Site on the same day cannot be unambiguously distinguished once imported into the CCC-DST app.

The only method of distinguishing two such Dives at a fundamental level is using the "id" property of the Dive within the ThinkSpatial app. However, this is only useful during the import process itself, as this number is not stored in either the Eye on the Reef database or the CCC-DST app. Even checking for doubled records in the ThinkSpatial exported data is imperfect, because two separate but seemingly identical Dives could, in theory, be collected on two separate tablets, and therefore imported in two separate processes that cannot easily be compared; however this edge case seems sufficiently unlikely that we ignore it for now.

As a result, checking to see whether a record already exists in the CCC-DST database requires:

1. Looking through all data records imported from the ThinkSpatial export files and seeing if any are "seemingly" doubled, while actually being distinct, and keeping track of those
2. For those that are not doubled, checking whether a record already exists in the CCC-DST database with all the same details, and if not adding it
3. For those that are doubled, tripled or more, checking whether the right number of records already exist in the CCC-DST database with all the same details, and if not, adding sufficient copies to create the right number.

### 2.4.7 Adding new records to the database

Because all cross-referencing to existing records is completed during the analysis to process the exported JSON, and by this point in the import process the relevant data has been encapsulated in the appropriate custom Types within the CCC-DST app, it is guaranteed to be in the appropriate format for inclusion in the mysql app database, and so this process is relatively straight-forward.

We simply add each component of the relevant custom type to a new "ContentValues", and then leverage the Android SQLLiteDatabase Insert method to insert it into the app sqlite database.

### *2.4.8 Updating the User Interface*

This process simply uses the same custom methods developed for displaying any other Control Program data, refreshing the display after the new data is loaded. This also allows the user to trigger a new Generate Worklist process to provide updated recommendations on which Sites should be culled next, given the data that was just imported.

# 3.0 THE DECISION SUPPORT TOOL

The COTS Control Centre Decision Support System is designed to support on-water operators making the day-to-day decisions required to implement the National COTS Control Program. The National COTS Control Program is designed around an ecologically-informed framework, which was outlined in the report *An ecologically-based operational strategy for COTS Control: Integrated decision making from the site to the regional scale* (Fletcher *et al.*, 2020).

That report lays out an explicit decision tree covering all the day-to-day ecologically-informed decisions required to operate the Program. At each decision point in the tree, data collected during previous control actions is used to select the next control action in an ecologically-informed manner. However, several of those decision points require complicated analysis of previously collected data, or the combination of multiple types of data in order to make the correct decision. The COTS Control Centre Decision Support System is designed to provide this analysis and combination of data using an on-water tablet system to support ecologically-informed decision-making at sea.

This section of this report provides some brief background, then details how the COTS Control Centre Decision Support System translates the formal decision tree process presented in Fletcher *et al.* (2020) into a software product that implements that decision tree.

## 3.1 Overview of an ecologically-informed underpinning for control

The ecologically-informed approach to COTS control outlined in the Fletcher *et al.* (2020) report is based on seven broad ecological principles: 1) prioritise Reefs for control (you can't manage everywhere at once, so prioritise your efforts); 2) cull entire Reefs (don't try to manage only parts of a reef because starfish can move and reinvade controlled areas); 3) prioritise Sites at Reefs (cull the places with the most COTS first to save the most coral); 4) cull Sites to below the Ecological Threshold (keep going until you've made a meaningful difference); 5) revisit Sites at Voyage intervals (don't revisit Sites so often that efficiency is reduced); 6) maintain surveillance after the Ecological Threshold is reached (once you've made an investment, maintain it); and 7) collect data as you go (the control program itself is the best source of data to inform decisions and further improve the control program). The COTS Control Centre Decision Support System is designed to address and support on-water operators in achieving each of these principles.

Based on these ecological and management drivers, there are a range of scales over which decisions need to be made by different decision makers in the system, from the entire GBR to the actions of individual divers. The decision tree process and the CCC-DSS are targeted at the decisions that are being made day-to-day on-board control vessels at sea. Specifically, the CCC-DSS aims to inform decision about how to carry out control actions once a Vessel arrives at a Reef, based on whether the Reef is in Intensive Control Mode, while COTS densities are above the Ecological Threshold, or Maintenance Mode, once COTS densities have been reduced below the Ecological Threshold. The decision trees and the CCC-DSS tools provide an ecologically-informed framework to address these decisions, a structure to ensure that the data to inform these decisions can be collected at the scales required, and a logical sequence to ensure that ecologically-meaningful outcomes are achieved.

The decision-making process may be structured as a “tree” that presents a flow of data collection, analysis, decision points, and resultant actions (Fig. 2). The tree structure ensures that the sequence of these events proceeds in a consistent order, while the fact that management continues until each Site is measured to drop below the relevant Ecological Threshold guarantees that the outcome is ecologically meaningful.

A broad outline of the process of following the decision tree is:

1. By the time it is leaving port, a Vessel will have an agreed Voyage Plan, which is determined in consultation with program managers at GBRMPA, defining which Reefs are to be visited during the Voyage, and in which order.
2. If the first Reef to be visited is a Maintenance Mode Reef that requires manta tow surveillance, the vessel travels to the Reef, manta tows the entire circumference of the Reef; if any sign of COTS is detected in any individual manta tow, the Reef is moved from Maintenance Mode to the top of the Intensive Control list; the most recent date of manta tow is updated and the vessel moves to the next Reef on the Voyage Plan.
3. If the first Reef to be visited is an Intensive Control Reef that has never been visited for control previously, or if it has been visited for control previously but the last manta tow at the Reef was performed more than 42 days ago, then the Vessel begins by performing a comprehensive manta tow around the circumference of the Reef. If no sign of COTS is found across any manta tow, the Reef is moved from Intensive Control mode to Maintenance Mode; the most recent date of manta tow is updated and the Vessel moves to the next Reef on the Voyage Plan.
4. If COTS are detected during the manta tow of the entire Reef, then any Site at which COTS were detected is targeted for control. This can be summarised as the rule “Dive a Site if one or more COTS or feeding scars are observed”. The Site with the highest number of COTS detections is controlled first.
5. Divers continue culling at all Sites on a Reef at which COTS were detected during the manta tow until either every such Site has been culled, in which case the Vessel moves to the next Reef on the Voyage Plan, or until it is time to return to port.
6. If all Sites on a Reef at which COTS were detected during the manta tow are culled and achieve CPUEs less than the Ecological Threshold, the Reef is moved from Intensive Control to Maintenance Mode, otherwise it remains in Intensive Control mode.
7. If the Reef being visited is an Intensive Control Reef that has been manta towed within the past 42 days, then no new manta tow is required, and divers begin culling: first, at any Site that was only partially completed during the previous Voyage; then at the Sites controlled during the previous Voyage for which the CPUE was above the Ecological Threshold, starting with the Site that produced the highest CPUE; and finally at any Sites that were not previously visited (due to the Voyage ending), but at which COTS were detected during the previous manta tow.

The COTS-DSS is designed to cover steps 2 – 7 of this process, by: 1) calculating whether a Reef meets the criteria for Intensive or Management mode; 2) advising when a manta tow is required based on the most recent manta tow date and the mode of the Reef; and 3) comparing the most recent cull or manta tow data for each Site to determine whether and in which order Sites should be culled.

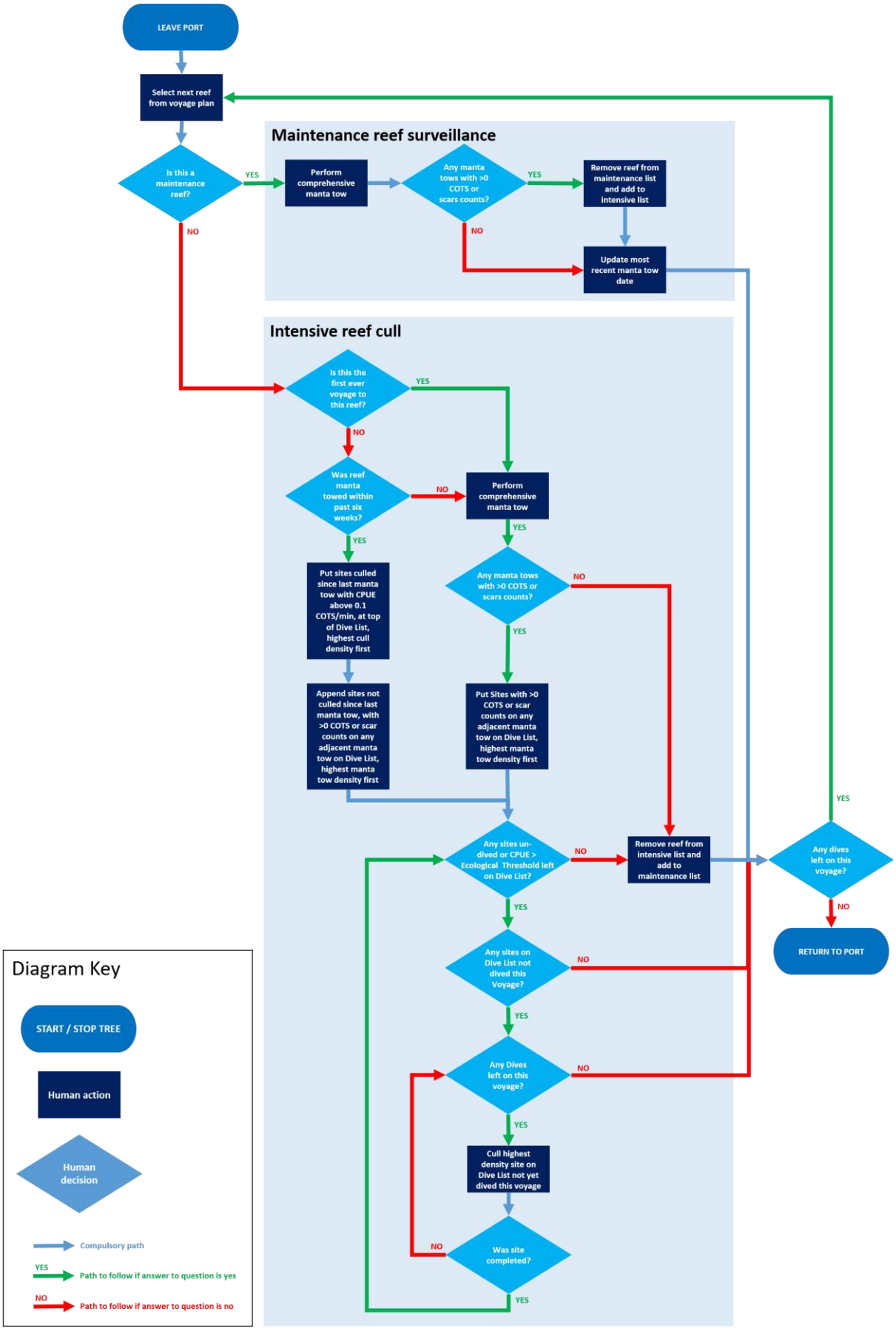



**Figure 8: The simplified on-water decision tree**

## 3.3 Implementing the Decision Tree

Although the decision tree presented in Figure 2 represents a logically complete summary of all the decisions that need to be made on-water in an ecologically-informed manner, in implementing it across a portfolio of interacting applications on the COTS Control Centre on-water hardware, we effectively break the tree up into several components, related to collecting different types of data, transferring that data between apps, and then implementing parts of the decision tree in order to guide on-water decision making.

On-water data collection is mediated on the COTS Control Centre Decision Support System tablets by apps created for GBRMPA by ThinkSpatial. Three data collection apps were created, with advice from the NESP IPM COTS Research Program, for collecting: 1) cull data; 2) manta tow data; and 3) Reef Health Information Survey (RHIS) data. In the first iteration of the CCC-DSS, the cull and manta tow data is used to guide on-water decision making; in future the RHIS data may also be used.

One key role of the CCC-DST is to collate this data from across the three separate data collection apps, incorporate it within the same decision support framework, and perform the necessary analysis to make it useful to on-water operators without requiring further manual assessment. This involves importing from the three data collection apps into the CCC-DST, performing georeferencing on the various data sources against each other and against the location of the Control Program Sites to spatially correlate them, and then comparing the data at each Site to guide ecologically-informed decisions.

At any specific point in time, only part of the decision tree may be required by the CCC-DST in order to inform a decision. At the same time, the decision tree presents a whole-of-voyage overview of the decisions being made, whereas the CCC-DST is likely to be used multiple times a day. This compartmentalisation of functionality, where some parts of the tree must be recalculated again every time new data becomes available, gives the implementation of the decision tree in the CCC-DST app a slightly different structure to the decision tree shown in Figure 2.

In practice, the decisions to be guided by the CCC-DST can be separated into five broad categories, the first a data processing step, the other four decisions to be made on-water:

1. How can Manta tows be associated with Sites?
2. Does this Reef meet the criteria for Intensive or Maintenance management mode?
3. Does the Reef need to be manta towed?
4. Does this Reef need to be culled?
5. If so, in what order should we cull the Sites?

Of these, the first and the last are by far the most complicated. However, implementing the middle three within the CCC-DST is important to ensure compliance with the ecologically-informed design of the National COTS Control Program. Below, we detail how each of these decisions is addressed within the CCC-DST.

### *3.3.1 Associating manta tows with Sites*

The entire framework of the National COTS Control Program is built around collecting the information required to ensured that key decisions are made with ecological information. A key component of that ecological information is provided by manta tows. In practice, however, manta tows occur *near* rather than *at* control Sites, so it is necessary to georeference them to control Sites before they can be used to inform decision making.

Georeferencing spatial data is a common process available in desktop Geographic Information Systems, but on-water, operators need to be able to determine which Sites each manta tow should be mapped against without relying on advanced processing or expensive software. Ideally, the process should be automated, so that data can feed directly into the CCC-DST.

In the CCC-DST, we leverage a hybrid approach in which the computationally expensive part of the process is pre-calculated ahead of time, so that manta tows can be quickly and accurately assigned to cull Sites as data from the ThinkSpatial surveillance app is imported into the CCC-DST.

For each Priority Reef for which GBRMPA provided predefined Sites:

1. A bounding box was defined buffered by 0.1 degrees around the maximum and minimum latitude and longitude of the Reef polygon
2. Within that bounding box, a raster grid of resolution 0.01 degrees was defined
3. At all points on that grid, the closest Site was calculated
4. The siteId from the siteTable was recorded at that raster pixel location in the grid
5. The raster grid was saved with header metadata defining the size and location of the raster grid, followed by the grid itself in .csv format

These grids provide lookup tables for each Reef, defining the information required to determine which Site is closest to any latitude and longitude within the bounding box of each Reef.

In the CCC-DST, when new surveillance data is imported, for each manta tow:

1. The Reef of the manta tow is extracted from the ThinkSpatial JSON data field
2. The lookup table for that Reef is loaded into the app memory
3. The start and stop latitude and longitude of the manta towline are used to estimate the mean latitude and longitude
4. That mean latitude and longitude is converted to a raster coordinate using the metadata header information in the lookup table
5. The nearest Site to the manta tow is read from the lookup table at the calculated raster coordinate

This system has the benefit of being very fast on-tablet, and much faster than a true georeferencing capability, but it has some constraints. At the moment, manta tows are assigned based on their median latitude and longitude only; in future the system could be refined to provide proportional attribution where manta tows cross Site polygon boundaries. The lookup table has a finite resolution, which is sufficient for the task it is being used for, but not as accurate as a vector based georeferencing capability. Perhaps most problematically,

the system only works for the Sites defined at the time the lookup tables were generated; as new Sites are added, lookup tables will have to be regenerated. This means that recommendations cannot currently be provided for new Sites defined during the current Voyage. As most Voyages are unlikely to define new Sites, the effect of this shortcoming is expected to be minor.

### *3.3.2 Assigning Reefs to Intensive or Maintenance mode*

Prior to any management taking place there, a Reef begins in Intensive Management mode, and may only be moved into Maintenance Mode if: 1) a comprehensive manta tow around the Reef detects no COTS or COTS Scars on any manta tow; or 2) at every Site for which COTS were detected during a manta tow, the most recent cull achieves a CPUEs less than the Ecological Threshold.

When the decision support function is run in the app, the CCC-DST:

1. looks up the most recent manta tow data for the Reef;
2. If it contains zero COTS or COTS Scars in every manta tow, the Reef is assigned to Maintenance Mode, otherwise:
3. It looks up the most recent Dive data for the Reef
4. For any Site at which a non-zero COTS or COTS scars manta tow was discovered it checks whether there has been a subsequent Dive:
    a. If not the Reef is assigned to Intensive Mode
    b. If so, it checks if the CPUE of the most recent Dive was greater than the Ecological Threshold of 0.04 COTS/minute bottom time:
        i. If so, the Reef is assigned to Intensive Mode
    c. If all Sites at which a non-zero COTS or COTS scars manta tow was discovered have since been culled and achieved a CPUE below the Ecological Threshold, the Reef is assigned to Maintenance Mode

It is important to note that this process calculates the ecologically-informed recommendation on whether a Reef meets the criteria to be transitioned from Intensive to Maintenance Mode. The formal decision to move a Reef from Intensive to Maintenance Mode must be made in conjunction with GBRMPA, so in this case, the app should be viewed as providing guidance requiring confirmation with GBRMPA. Because of this, the recommendation by the app that a Reef meets the criteria for Maintenance Mode can be overridden in the app to provide recommendations for culling even when the app believes the Reef meets the criteria to stop culling.

### *3.3.3 Informing the decision to manta tow*

The ecologically-informed approach to COTS control outlined in the Fletcher *et al.* (2020) leverages manta tows for two reasons:

1. To ensure that Reefs that have been moved to Maintenance Mode remain below the Ecological Threshold;
2. To guide the order in which Sites at Intensive Mode Reefs are culled

Each of these tasks requires different re-manta tow frequencies, nominally:

1. Maintenance Mode Reefs: once every six months;
2. Intensive Mode Reefs: once every 42 days

When the decision support function is run in the app, the CCC-DST:

1. Tests whether the Reef is in Intensive or Maintenance Mode
2. Looks up the most recent manta tow data for the Reef;
3. Calculates the number of days between the current date and the most recent manta tow
4. If the Reef is a Maintenance Mode Reef, recommends a comprehensive manta tow if the most recent manta tow was more than 183 days ago
5. If the Reef is an Intensive Control Reef, recommends a comprehensive manta tow if the most recent manta tow was more than 42 days ago.

Although these frequencies are hard coded in the current version of the app, they are coded as a global variable and easily refined with input from GBRMPA as we learn more about the effectiveness of manta tows in the Control Program. Future versions of the CCC-DST may refine them automatically using artificial intelligence and adaptive management principles.

### *3.3.4 Informing the decision to cull*

If the Reef has been determined to be in Intensive Management Mode, and it is the start of a Voyage, then under the current structure of the decision tree, some Sites will need to be culled. However, although the question of whether or not to cull may seem redundant in the context of the decision tree, which conceptually is used only once per Voyage, given the following question about in which order to cull Sites, it is very important in the context of the Decision Support Tool, which could be used multiple times every day.

In this context, the important parameter of the decision tree process is the “Site revisitation frequency”. The current Control Program aims to revisit Sites once per voyage, roughly every 12 days, an operational approximation based on expert opinion about how soon after culling Divers can reliably distinguish previously culled COTS from unculled COTS at a Site, and how quickly cryptic COTS hidden within the reef matrix during one control Dive emerge and become available to be culled on the next Dive.

This means that, each time the CCC-DST is run:

1. After we have checked that the Reef is in Intensive Management Mode; we
2. Check whether any of the Sites that need to be culled were last culled more than 12 days ago.

In practice, this step is combined with the following step determining in which order to cull Sites.

Again, although the “Site revisitation frequency” is hard coded in the current version of the app, it is coded as a global variable and easily refined as we learn more about the effectiveness of

Site revisitation frequency in the Control Program. Future versions of the CCC-DST may refine it automatically using artificial intelligence and adaptive management principles.

### *3.3.5 Informing the order in which Sites are culled*

This component is where the bulk of the work of the CCC-DST takes place.

The output of this process is a list of the Sites to be culled, in order, based on the underlying principles of the COTS decision support tree:

1. Cull the Sites with the highest COTS density first
2. Once you start culling a Site, keep culling it intensively until it is reduced below the Ecological Threshold
3. Use regular manta tows to gain a whole-of-Reef intelligence and reset your target Sites every four Voyages.

In practice, this is implemented by, each time the CCC-DST is run, the system:

1. Looks up the most recent Manta tow for the Reef
2. Looks up the most recent Dives for each Site on the Reef
3. For each Site at which the most recent event was a Dive:
    a. Check the CPUE of the most recent Dive, and if the CPUE was greater than the Ecological Threshold
    b. Check the date of the most recent Dive, calculate the number of days since the Dive, and if it was greater than the Site Revisitation Frequency
    c. Add the Site to the recommended Cull Worklist
4. For each Site at which the most recent event was a Manta:
    a. Check whether the 2 – 4 mantas associated with the Site during the most recent Manta tow detected any COTS or COTS Scars
    b. If so, add the Site to the recommended Manta Worklist
5. Sort the recommended Cull Worklist by highest CPUE first
6. Add the sorted recommended Cull Worklist to the recommended Worklist
7. Sort the recommended Manta Worklist by highest COTS detections first
8. Append the sorted Manta Worklist to the recommended Worklist
9. Present the Worklist to the user

# 4.0 SUMMARY

This report details the design philosophy and implementation of the COTS Control Centre Decision Support Tool (CCC-DST) component of the COTS Control Centre Decision Support System (CCC-DSS). The CCC-DSS is a hardware and software solution designed by CSIRO to underpin ecologically-informed data collection and on-water management decision making in the National COTS Control Program. It consists of a fleet of 32 Samsung Galaxy Tab Active2 Wi-Fi tablets; an Enterprise Management System powered by Samsung Knox Manage; three data collection apps developed for GBRMPA by ThinkSpatial; and three components developed as part of the NESP COTS IPM research program to inform on-water decision making, and explore and share data between applications and tablets.

# REFERENCES

Bruno, J. F., & Selig, E. R. (2007). Regional Decline of Coral Cover in the Indo-Pacific: Timing, Extent, and Subregional Comparisons. *PLoS ONE, 2*(8), e711. doi:10.1371/journal.pone.0000711

Claar, D. C., Szostek, L., McDevitt-Irwin, J. M., Schanze, J. J., & Baum, J. K. (2018). Global patterns and impacts of El Niño events on coral reefs: A meta-analysis. *PLoS ONE, 13*(2), e0190957. doi:10.1371/journal.pone.0190957

De'ath, G., Fabricius, K. E., Sweatman, H., & Puotinen, M. (2012). The 27-year decline of coral cover on the Great Barrier Reef and its causes. *Proceedings of the National Academy of Sciences, 109*(44), 17995-17999. doi:10.1073/pnas.1208909109

Firth, S., & McKenzie, C. (2015). *Crown-of-thorns starfish (COTS) control program : Voyage data - Catch per Unit of Effort (CPUE), Eradicated COTS size and coral cover from July 2013 - Jan 2015 (AMPTO)*. Retrieved from: http://eatlas.org.au/geonetwork/srv/eng/metadata.show?uuid=fe100cb3-9b29-4e6b-a8e1-0434631064fa&currTab=complete

Fletcher, C. S., Bonin, M. C., & Westcott, D. A. (2020). *An ecologically-based operational strategy for COTS Control: Integrated decision making from the site to the regional scale*. Retrieved from Cairns:

Fletcher, C. S., & Westcott, D. A. (2016). *Strategies for Surveillance and Control: Using Crown-of-Thorns Starfish management program data to optimally distribute management resources between surveillance and control. Report to the National Environmental Science Programme. Reef and Rainforest Research Centre Limited*. Retrieved from Cairns:

Gardner, T. A., Côté, I. M., Gill, J. A., Grant, A., & Watkinson, A. R. (2003). Long-Term Region-Wide Declines in Caribbean Corals. *Science, 301*(5635), 958-960. doi:10.1126/science.1086050

Hoegh-Guldberg, O. (1999). Climate change, coral bleaching and the future of the world's coral reefs. *Mar. Freshwater Res., 50*(8), 839. doi:10.1071/mf99078

Hughes, T. P. (2003). Climate Change, Human Impacts, and the Resilience of Coral Reefs. *Science, 301*(5635), 929-933. doi:10.1126/science.1085046

Hughes, T. P., Anderson, K. D., Connolly, S. R., Heron, S. F., Kerry, J. T., Lough, J. M., . . . Wilson, S. K. (2018a). Spatial and temporal patterns of mass bleaching of corals in the Anthropocene. *Science, 359*(6371), 80-83. doi:10.1126/science.aan8048

Hughes, T. P., Kerry, J. T., & Simpson, T. (2018b). Large-scale bleaching of corals on the Great Barrier Reef. *Ecology, 99*(2), 501-501. doi:10.1002/ecy.2092

Moutardier, G., Gereva, S., Mills, S. C., Adjeroud, M., Beldade, R., Ham, J., . . . Dumas, P. (2015). Lime Juice and Vinegar Injections as a Cheap and Natural Alternative to Control COTS Outbreaks. *PLoS ONE, 10*(9), 16.

Pearson, R. G. (1981). Recovery and Recolonization of Coral Reefs. *Marine Ecology Progress Series, 4*, 105-122. doi:10.3354/meps004105

Pratchett, M. S., Caballes, C. F., Rivera-Posada, J. A., & Sweatman, H. P. A. (2014). Limits to Understanding and Managing Outbreaks of Crown-of-Thorns Starfish (*Acanthaster* spp.). *Oceanography and Marine Biology: An Annual Review, 52*, 133-200. doi:doi:10.1201/b17143-4

Rivera-Posada, J. A., Pratchett, M., Cano-Gómez, A., Arango-Gómez, J. D., & Owens, L. (2011). Injection of Acanthaster planci with thiosulfate-citrate-bile-sucrose agar (TCBS). I. Disease induction. *Diseases of Aquatic Organisms, 97*(2), 85-94. doi:10.3354/dao02401

Westcott, D. A., & Fletcher, C. S. (2018). *How Effective Are Management Responses In Controlling Crown-of-Thorns Starfish and their Impacts On The Great Barrier Reef? Report to the National Environmental Science Program. Reef and Rainforest Research Centre Limited*. Retrieved from Cairns:

Westcott, D. A., Fletcher, C. S., Kroon, F. J., Babcock, R. C., Plagányi, E. E., Pratchett, M. S., & Bonin, M. C. (2020). Relative efficacy of three approaches to mitigate Crown-of-Thorns Starfish outbreaks on Australia's Great Barrier Reef. *Scientific Reports, 10*(1), 12594. doi:10.1038/s41598-020-69466-1

Yamaguchi, M. (1986). Acanthaster planci infestations of reefs and coral assemblages in Japan: a retrospective analysis of control efforts. *Coral Reefs, 5*(1), 23-30. doi:10.1007/bf00302168

# APPENDIX A: THINKSPATIAL JSON STRUCTURES

## A.1 culldata.db – culldata Table – json field


```
{
    "bottomTime": 120,
    "cohorts": [
        0,
        1,
        6,
        5
    ],
    "cpue": 0.1,
    "cullzone": {
        "feature": {
            "geometry": {
                "coordinates": [
                    [
                        [
                            147.0723769,
                            -18.6165982
                        ],
```

… however many [Longitude, Latitude] pairs are required to define the cull polygon


```
            ],
                "type": "Polygon"
            },
            "properties": {
                "id": 11494,
                "reef_id": 644,
                "reefname": "John Brewer Reef (18-075)",
                "sitename": "JOH_18-075_35"
            },
            "type": "Feature"
        },
        "id": 11494,
        "name": "JOH_18-075_35",
        "reefId": 644
    },
    "depth": "5",
    "divedate": "2020-11-06T14:00:00.000Z",
    "reef": {
        "feature": {
            "geometry": {
                "coordinates": [
                    [
                        [
                            147.040407,
                            -18.65251
```

```
                        ],
… however many [Longitude, Latitude] pairs are required to define the reef polygon
                    ]
                ],
                "type": "Polygon"
            },
            "properties": {
                "id": 644,
                "label": "18-075",
                "name": "John Brewer Reef (18-075)",
                "priority": true
            },
            "type": "Feature"
        },
        "id": 644,
        "name": "John Brewer Reef (18-075)",
        "priority": true
    },
    "voyage": {
        "end": "2020-11-17",
        "id": 529,
        "start": "2020-11-03",
        "title": "47",
        "vessel": "Pacific Marine Group"
    }
}
```


## A.2 surveillance.db – surveillance Table – json field


```
{
    "cotsSeen": "0",
    "cullzones": [
        {
            "feature": {
                "geometry": {
                    "coordinates": [
                        [
                            [
                                147.02994,
                                -18.6362887
                            ],
… however many [Longitude, Latitude] pairs are required to define the cull polygon
                        ]
                    ],
                    "type": "Polygon"
                },
                "properties": {
                    "id": 2304,
```

```
                    "reef_id": 644,
                    "reefname": "John Brewer Reef (18-075)",
                    "sitename": "JOH_18-075_18"
                },
                "type": "Feature"
            },
            "id": 2304,
            "name": "JOH_18-075_18",
            "reefId": 644
        },
```


… how every many cullZones are intersected by the manta tow


```
    ],
    "deadCoralCat": "0",
    "hardCoralCat": "1-",
    "id": 1,
    "reef_id": 644,
    "scarsSeen": "a",
    "softCoralCat": "1-",
    "submitted": false,
    "towAvgSpeed": 7.24,
    "towDate": "2020-11-23T22:10:41.299Z",
    "towDistance": 235.3,
    "towLine": {
        "geometry": {
            "coordinates": [
                [
                    147.0297746,
                    -18.63267
                ],
```


… however many [Longitude, Latitude] pairs are required to define the tow line


```
            ],
            "type": "LineString"
        },
        "properties": {
        },
        "type": "Feature"
    },
    "voyage": {
        "end": "2020-12-02",
        "id": 530,
        "start": "2020-11-18",
        "title": "48",
        "vessel": "Pacific Marine Group"
    }
}
```

## A.3 rhisdata.db – rhis Table – json field


```
{
    "Benthos Observations": [
        {
            "Coral Rubble": 0,
            "Live Coral": 20,
            "Live Coral Rock": 78,
            "Macroalgae": 1,
            "Observations": "Benthos (%)",
            "Recently Dead Coral": 1,
            "Sand": 0,
            "ValueLookup": null,
            "ValueType": "PROPORTION"
        }
    ],
    "Bleaching Cause": [
        {
            "Bleaching Cause": 1,
            "Observations": "Likely Cause of Bleaching"
        }
    ],
    "Breakage Observations": [
        {
            "Observations": "Proportion of Coral Cover Affected (%)",
            "Total": 1,
            "ValueLookup": null,
            "ValueType": "DISCRETE_PCNT"
        }
    ],
    "Coral Bleaching Observations": [
        {
            "Branching": 0,
            "Bushy": 0,
            "Encrusting": 0,
            "Massive": 0,
            "Mushroom": 0,
            "Observations": "Proportion of the Corals that are bleached (%)",
            "Plate/Table": 0,
            "Soft": 0,
            "ValueLookup": null,
            "ValueType": "DISCRETE_PCNT",
            "Vase/Foliose": 0
        },
        {
            "Branching": 0,
            "Bushy": 0,
            "Encrusting": 0,
            "Massive": 0,
```

```
            "Mushroom": 0,
            "Observations": "Severity",
            "Plate/Table": 0,
            "Soft": 0,
            "ValueLookup": "4",
            "ValueType": "LOOKUP",
            "Vase/Foliose": 0
        }
    ],
    "Coral Disease Cover": [
        {
            "Black Band": 0,
            "Brown Band": 0,
            "Observations": "Proportion of Coral Cover Affected (%)",
            "Other disease/tumors": 0,
            "ValueLookup": null,
            "ValueType": "PERCENT",
            "White Syndromes": 0
        }
    ],
    "Coral Disease Observations": [
        {
            "Branching": 0,
            "Bushy": 0,
            "Encrusting": 0,
            "Massive": 0,
            "Mushroom": 0,
            "Observations": "Black Band (colonies affected)",
            "Plate/Table": 0,
            "Soft": 0,
            "ValueLookup": null,
            "ValueType": "NUMBER",
            "Vase/Foliose": 0
        },
        {
            "Branching": 0,
            "Bushy": 0,
            "Encrusting": 0,
            "Massive": 0,
            "Mushroom": 0,
            "Observations": "Brown Band (colonies affected)",
            "Plate/Table": 0,
            "Soft": 0,
            "ValueLookup": null,
            "ValueType": "NUMBER",
            "Vase/Foliose": 0
        },
        {
            "Branching": 0,
            "Bushy": 0,
```

```
            "Encrusting": 0,
            "Massive": 0,
            "Mushroom": 0,
            "Observations": "White Syndromes (colonies affected)",
            "Plate/Table": 0,
            "Soft": 0,
            "ValueLookup": null,
            "ValueType": "NUMBER",
            "Vase/Foliose": 0
        },
        {
            "Branching": 0,
            "Bushy": 0,
            "Encrusting": 0,
            "Massive": 0,
            "Mushroom": 0,
            "Observations": "Other disease/tumors (colonies affected)",
            "Plate/Table": 0,
            "Soft": 0,
            "ValueLookup": null,
            "ValueType": "NUMBER",
            "Vase/Foliose": 0
        }
    ],
    "Coral Observations": [
        {
            "Branching": 10,
            "Bushy": 40,
            "Encrusting": 5,
            "Massive": 0,
            "Mushroom": 0,
            "Observations": "Proportion of coral cover (live and recently dead)",
            "Plate/Table": 30,
            "Soft": 5,
            "ValueLookup": null,
            "ValueType": "PROPORTION",
            "Vase/Foliose": 10
        },
        {
            "Branching": 0,
            "Bushy": 1,
            "Encrusting": 0,
            "Massive": 0,
            "Mushroom": 0,
            "Observations": "Proportion of the above that is recently dead",
            "Plate/Table": 1,
            "Soft": 0,
            "ValueLookup": null,
            "ValueType": "DISCRETE_PCNT",
            "Vase/Foliose": 0
```

```
    }
],
"Coral Predation Observations": [
    {
        "Branching": 0,
        "Bushy": 0,
        "Encrusting": 0,
        "Massive": 0,
        "Mushroom": 0,
        "Observations": "COTS (number of scars)",
        "Plate/Table": 1,
        "Soft": 0,
        "ValueLookup": null,
        "ValueType": "NUMBER",
        "Vase/Foliose": 0
    },
    {
        "Branching": 0,
        "Bushy": 0,
        "Encrusting": 0,
        "Massive": 0,
        "Mushroom": 0,
        "Observations": "Drupella (number of scars)",
        "Plate/Table": 0,
        "Soft": 0,
        "ValueLookup": null,
        "ValueType": "NUMBER",
        "Vase/Foliose": 0
    }
],
"Macroalgae Observations": [
    {
        "Entangled/Mat-like": 30,
        "Filamentous": 30,
        "Leafy/Fleshy": 40,
        "Observations": "Proportion Total Macroalgae Cover (%)",
        "Slime": 0,
        "Tree/Bush-like": 0,
        "ValueLookup": null,
        "ValueType": "PROPORTION"
    },
    {
        "Entangled/Mat-like": 1,
        "Filamentous": 1,
        "Leafy/Fleshy": 1,
        "Observations": "Height (cm)",
        "Slime": 0,
        "Tree/Bush-like": 0,
        "ValueLookup": "1",
        "ValueType": "LOOKUP"
```

```
        }
    ],
    "Movement Cause": [
        {
            "Coral Disease Observations": 1,
            "Observations": "Algae Present",
            "Predator Observations": 1,
            "Recent Coral Damage": 1
        }
    ],
    "Predator Observations": [
        {
            "COTS": 1,
            "Drupella": 0,
            "Observations": "Proportion of Coral Cover Affected (%)",
            "ValueLookup": null,
            "ValueType": "PERCENT"
        },
        {
            "COTS": 0,
            "Drupella": null,
            "Observations": "Juveniles",
            "ValueLookup": null,
            "ValueType": "NUMBER"
        },
        {
            "COTS": 0,
            "Drupella": 0,
            "Observations": "Adults",
            "ValueLookup": null,
            "ValueType": "NUMBER"
        }
    ],
    "Recent Coral Damage": [
        {
            "Branching": 0,
            "Bushy": 2,
            "Encrusting": 0,
            "Massive": 0,
            "Mushroom": 0,
            "Observations": "Colonies Affected",
            "Plate/Table": 1,
            "Soft": 0,
            "ValueLookup": null,
            "ValueType": "NUMBER",
            "Vase/Foliose": 0
        },
        {
            "Branching": 0,
            "Bushy": 2,
```

```
            "Encrusting": 0,
            "Massive": 0,
            "Mushroom": 0,
            "Observations": "Severity",
            "Plate/Table": 2,
            "Soft": 0,
            "ValueLookup": "9",
            "ValueType": "LOOKUP",
            "Vase/Foliose": 0
        },
        {
            "Branching": 0,
            "Bushy": 6,
            "Encrusting": 0,
            "Massive": 0,
            "Mushroom": 0,
            "Observations": "Possible Cause",
            "Plate/Table": 4,
            "Soft": 0,
            "ValueLookup": "8",
            "ValueType": "LOOKUP",
            "Vase/Foliose": 0
        }
    ],
    "Rubbish Observations": [
        {
            "Fishing Line": 0,
            "Netting": 0,
            "Observations": "Pieces",
            "Other": 0,
            "Plastic": 0,
            "Rope": 0,
            "ValueLookup": null,
            "ValueType": "NUMBER"
        }
    ],
    "is_archived": "N",
    "static": [
        {
            "additionalinfo": "",
            "airtemp": 29,
            "algalbloom": "N",
            "aspect": "SW",
            "cotscull": 4125,
            "depth": 3,
            "email": "karen.eigeland@hotmail.com",
            "floodplume": "N",
            "gps": "147.26948, -18.756930",
            "habitat": "Crest",
            "observer": 1128,
```

```
            "organisation": "Pacific Marine Group",
            "phone": null,
            "prm": null,
            "reef": 725,
            "secchi": null,
            "sheetno": 3,
            "sheetof": 3,
            "site": null,
            "surveydate": "26-11-2020",
            "surveytime": "21:00",
            "surveytype": "POINT SURVEY",
            "swimtype": "SNORKEL",
            "tide": "H",
            "vessel": 205,
            "visibility": ">10m",
            "watertempdeep": 24,
            "watertempshallow": 26,
            "zonetype": 6
        }
    ],
    "surveyid": 0,
    "type": "REEF_HEALTH",
    "version": "2.1.0"
}
```

# APPENDIX B: SOURCE CODE

## B.1 MainActivity.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0;

import android.Manifest;
import android.content.Context;
import android.content.pm.PackageManager;
import android.database.Cursor;
import android.os.Bundle;

import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
import androidx.loader.app.LoaderManager;
import androidx.loader.content.CursorLoader;
import androidx.loader.content.Loader;
import androidx.appcompat.app.AppCompatActivity;

import android.util.TimingLogger;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;

import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.model.LatLng;

import java.util.ArrayList;
import java.util.Hashtable;
import java.util.List;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.ReefEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.ReefPolygonsEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.SitePolygonsEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.SiteEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.VesselEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.VoyageEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.DiveEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.MantaEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.RhisEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.model.ImplementDecisionTreeAtReef;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Reef;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.ReefPolygon;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.ReefPolygonPoint;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Site;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.SiteList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.SitePolygon;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.SitePolygonList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.SitePolygonPoint;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.SitePolygonPointList;
```

```
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Vessel;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Voyage;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Dive;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.DiveList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Manta;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.MantaList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Rhis;

;

// PURPOSE
//
// This activity:
//
//      1. lets the user select which Reef on the GBR to analyse based on the ecologically-
// informed decision tree outlined in "An ecologically-based operational strategy for COTS Control:
// Integrated decision making from the site to the regional scale" [1].
//
//      2. displays for the user the latest data available for the selected Reef, including:
//          a) the latest manta tow data
//          b) the latest site density estimate
//          c) the current status of each site (above or below ecological threshold)
//          d) a list of the recommended order in which to visit each site
//
// The user selects the reef to analyse based on:
//      1. pinch-and-zoom on Google map
//      2. GPS selected by pin button overlaid on map
//      3. text based reef name search in floating search bar
//
// The app allows the user to review the most up-to-date data for the current Reef, notifies the
// Decision Maker when it is time to manta tow the Reef, highlights which Sites the Decision
// Support Tool recommends diving on next, and allows the Decision Maker to manually override
// the Decision Support Tool for actions at a specific Site.
//
// Behind the scenes, the activity also updates the latest data from the GBRMPA data collection
// apps. A separate service checks for nearby COTS Control Tablets to automatically sync the data
// from those tablets to this tablet.
//
// PHILOSOPHY
//
// The app can be viewed as three key components:
//
//     1. A component that reads stored Control Program data from the app's sqlite database file
//
//     2. A component that runs the relevant subset of that data through the COTS Decision Tree
//        and provides recommendations on when to manta tow or which Sites to cull next
//
//     3. A component that display the data and recommendations using a map interface
//
// The COTS Control Program data is central to each of these components, and how it is stored and
```

```
// interacted with is a core part of the tool development.
//
// Database Tables
//
// The data is stored in an sqlite database within the app. The database is structured based on the
// physical structure of the system, and contains components related to geographic features and
// management actions. The Control Program operates at the intersection of these components, and
// so the way they are structured is important to the operation of the Decision Support Tool.
//
// Geographic Features: The database contains Tables for storing two types of geographic features:
// Reefs and Sites. Although Sites are defined by the Control Program, once they are defined they
// are fixed. As such, the Site table should also remain static and not contain any information
// related to management, other than the addition of new Sites.
//
// Management Actions: The database contains Tables for storing five types of management-related
// information: Vessels, Voyages, Dives (cull dives), Mantas (manta tows), and RHIS (Reef Health
// Information Surveys). These characteristics reflect management actions: Dives report how many
// COTS were culled, Voyages report dates that they occurred etc. As more control actions continue,
// more items get added to these Tables.
//
// Database Table Relationships
//
// Vessels are the basic unit of management.
//
// Voyages refer to a specific Vessel, and take place over a certain date period. They are not
// associated with a given Reef directly, rather the Dives or Mantas that take place during the
// Voyage are assigned to a Site or Reef, as below.
//
// Dives take place on a Voyage and at a Site and take place on a certain Date.
//
// Mantas take place on a Voyage and at a "Nearest Site", and take place on a certain Date.
// Technically, Mantas take place at a Reef, and we may want to change this structure to reflect
// that because, ultimately, this is a derived relationship rather than a fundamental one, because
// Mantas are not strictly contained within Sites. This will become more important if we want to
// introduce a maximum distance from a Site that a Manta can be
// assigned to it, but for now this works.
//
// RHIS take place on a Voyage and at a Site and take place on a certain Date.
//
// Data Types
//
// The data from the database Tables is loaded into custom types within the app. These types are
// based on similar principles to the Tables.
//
// In addition to the data types themselves, we define several types of Lists of data types. This
// is purely to enable sensible list-based search options - e.g. a function for returning the
// most recently visited Site from a list of Sites, based on the latest Dive, RHIS and/or nearby
// Manta tows.
//
//
```

```
// IMPLEMENTATION
//
//
// REFERENCES
//
// [1]   Fletcher C. S., Bonin M. C, Westcott D. A.. (2020) An ecologically-based operational
// strategy for COTS Control: Integrated decision making from the site to the regional scale.
// Reef and Rainforest Research Centre Limited, Cairns (65pp.).
//
// TODO: SEQUENCE DATA LOADING TO IMPROVE UI EXPERIENCE
//
// TODO: INVESTIGATE USING ONE SUPER-LOADER TO LOAD EVERYTHING NEEDED AT ONCE

// Note that AppCompatActivity extends FragmentActivity
public class MainActivity extends AppCompatActivity implements
        LoaderManager.LoaderCallbacks<Cursor>,
        DisplayMap.DisplayMapFragmentMarkerTouchListener,
        DisplayMap.DisplayMapFragmentMapTouchListener,
        DisplayMap.DisplayMapFragmentReadyListener,
        DisplayInfo.DisplayInfoFragmentLoadDataButtonPressListener,
        DisplayInfo.DisplayInfoFragmentGenerateWorkplanButtonPressListener,
        DisplayInfo.DisplayInfoFragmentReadyListener {

    // Define loader IDs
    private static final int ID_LOAD_REEF_TABLE = 100;
    private static final int ID_LOAD_REEFPOLYGONS_TABLE = 150;
    private static final int ID_LOAD_SITE_TABLE = 200;
//    private static final int ID_LOAD_SITEID_FROM_SITENAME = 201;
    private static final int ID_LOAD_SITEPOLYGONS_TABLE = 250;
    private static final int ID_LOAD_VESSEL_TABLE = 300;
    private static final int ID_LOAD_VOYAGE_TABLE = 400;
    private static final int ID_LOAD_DIVE_TABLE = 500;
    private static final int ID_LOAD_MANTA_TABLE = 600;
    private static final int ID_LOAD_RHIS_TABLE = 700;

    private Hashtable<Integer,Integer> cursorsStillLoadingData = new Hashtable<Integer,Integer>();

    private static int mapDisplayType;
    private static final int MAP_DISPLAY_ALL_REEFS_WITH_VOYAGES = 100;
    private static final int MAP_DISPLAY_REEF_WITH_SITES = 200;

// Define ArrayLists of the fundamental data types associated with the Control Program data
    public static List<Reef> reefList = new ArrayList<Reef>();
    public static List<ReefPolygonPoint> reefPolygonPointsList = new ArrayList<ReefPolygonPoint>();
    public static ReefPolygon reefPolygon;
    public static SiteList siteList = new SiteList();
    public static SitePolygonPointList sitePolygonPointsList = new SitePolygonPointList();
    public static SitePolygonList sitePolygonsList = new SitePolygonList();
    public static List<Vessel> vesselList = new ArrayList<Vessel>();
    public static List<Voyage> voyageList = new ArrayList<Voyage>();
```

```
    public static DiveList diveList = new DiveList();
    public static MantaList mantaList = new MantaList();
    public static List<Rhis> rhisList = new ArrayList<Rhis>();

    public static Reef selectedReef;

    public Context context;

    private DisplayMap displayMapFragment;
    private DisplayInfo displayInfoFragment;

    public static GoogleMap mainMap;
    private boolean firstMap = true;

    private String siteName;

//  These are working variables that are just used during the coding and debugging of the
//  application. In the long-run they will be replaced with dynamically created variables

    // Latitude and longitude of Cairns
    private static double CAIRNS_LATITUDE = -16.9186;
    private static double CAIRNS_LONGITUDE = 145.7781;
    private static LatLng cairnsLatLng = new LatLng( CAIRNS_LATITUDE, CAIRNS_LONGITUDE );

    public static double ecologicalThresholdCPUE = 0.1;
    public static int ecologicalThresholdMantaCOTS = 0;
    public static String ecologicalThresholdMantaScars = "a";

    public static int daysToMantaTowMaintenanceReef = 183 /* default 183 - six months */;
    public static int daysToMantaTowIntensiveControlReef = 42 /* default 42 - every fourth 10 - 12 day
voyage */;
    public static int daysToCullSiteAtIntensiveControlReef = 10; /* default 10 - every 10 - 12 voyage */

    public static TimingLogger loaderTiming = new TimingLogger( "CCC_LOADER", "loadingTimingStart");

//
//
// MAIN onCreate function
//
//

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        //Remove title bar
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);

        //Remove notification bar
        this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

```
        // Set the theme to the app theme to replace the splash screen theme
        setTheme( R.style.AppTheme );

        // First, call the super.onCreate method
        super.onCreate( savedInstanceState );

        context = getApplicationContext();

        checkPermission();

        // Set the content view
        setContentView( R.layout.activity_main );

        // Initialize a series of loaders. If a loader doesn't already exist, one is created and started.
        // and (if the activity/fragment is currently started) starts. We use restartLoader rather
        // than initLoader (which would reuse the old loader if it already existed) because there
        // seems to be a bug in the SupportLoaderManager class that prevents the cursor data
        // persisting with an orientation change.

        mapDisplayType = MAP_DISPLAY_ALL_REEFS_WITH_VOYAGES;

        //TODO: Consider removing this and the ReefInfo fragment as dynamically loaded fragments
        // and just show and hide them as necessary
        //
        // Set up the display_map fragment
        getSupportFragmentManager().beginTransaction()
                .setReorderingAllowed(true)
                .add( R.id.display_map_fragment_container_view, new DisplayMap(), null )
                .add( R.id.display_info_fragment_container_view, new DisplayInfo(), null )
                .commit();

        // Set up the display_info fragment
//        getSupportFragmentManager().beginTransaction()
//                .setReorderingAllowed(true)
//                .add( R.id.display_info_fragment_container_view, new DisplayInfo(), null )
//                .commit();

        getSupportLoaderManager().initLoader(ID_LOAD_REEF_TABLE, null, this);


    }

//
//
// onCreateLoader function
//
//
    @Override
    public Loader<Cursor> onCreateLoader(int loaderId, Bundle bundle) {
```

```
        switch ( loaderId ) {

            case ID_LOAD_REEF_TABLE:

                cursorsStillLoadingData.put( ID_LOAD_REEF_TABLE, 1 );

                return new CursorLoader(this,
ReefEntry.CONTENT_URI.buildUpon().appendPath("controlled").build(), null, null, null, null);

            case ID_LOAD_REEFPOLYGONS_TABLE:

                cursorsStillLoadingData.put( ID_LOAD_REEFPOLYGONS_TABLE, 1 );

                return new CursorLoader(this, ReefPolygonsEntry.CONTENT_URI, null,
ReefPolygonsEntry.REEF_POLYGONS_TABLE_COLUMN_REEF_ID + "=" + Integer.toString( selectedReef.getReefId() ) ,
null, null);

            case ID_LOAD_SITE_TABLE:

                cursorsStillLoadingData.put( ID_LOAD_SITE_TABLE, 1 );

                return new CursorLoader(this, SiteEntry.CONTENT_URI, null,
SiteEntry.SITE_TABLE_COLUMN_REEF_ID + "=" + Integer.toString( selectedReef.getReefId() ) , null, null);

//            case ID_LOAD_SITEID_FROM_SITENAME:
//
//                cursorsStillLoadingData.put( ID_LOAD_SITEID_FROM_SITENAME, 1 );
//
//                return new CursorLoader(this,
SiteEntry.CONTENT_URI.buildUpon().appendPath("where/sitename").build(), null,
SiteEntry.SITE_TABLE_COLUMN_SITE_NAME + "=" + siteName , null, null);

            case ID_LOAD_SITEPOLYGONS_TABLE:

                cursorsStillLoadingData.put( ID_LOAD_SITEPOLYGONS_TABLE, 1 );

                return new CursorLoader(this, SitePolygonsEntry.CONTENT_URI, null,
SiteEntry.SITE_TABLE_COLUMN_REEF_ID + "=" + Integer.toString( selectedReef.getReefId() ) , null, null);

            case ID_LOAD_VESSEL_TABLE:

                cursorsStillLoadingData.put( ID_LOAD_VESSEL_TABLE, 1 );

                return new CursorLoader(this, VesselEntry.CONTENT_URI, null, null, null, null);

            case ID_LOAD_VOYAGE_TABLE:

                cursorsStillLoadingData.put( ID_LOAD_VOYAGE_TABLE, 1 );

                return new CursorLoader(this, VoyageEntry.CONTENT_URI, null,
SiteEntry.SITE_TABLE_COLUMN_REEF_ID + "=" + Integer.toString( selectedReef.getReefId() ), null, null);

            case ID_LOAD_DIVE_TABLE:
```

```
                cursorsStillLoadingData.put( ID_LOAD_DIVE_TABLE, 1 );

                return new CursorLoader(this,
DiveEntry.CONTENT_URI.buildUpon().appendPath("where").appendPath("reef").build(), null,
SiteEntry.SITE_TABLE_NAME + "." + SiteEntry.SITE_TABLE_COLUMN_REEF_ID + "=" + Integer.toString(
selectedReef.getReefId() ), null, null);

            case ID_LOAD_MANTA_TABLE:

                cursorsStillLoadingData.put( ID_LOAD_MANTA_TABLE, 1 );

                return new CursorLoader(this,
MantaEntry.CONTENT_URI.buildUpon().appendPath("where").appendPath("reef").build(), null,
SiteEntry.SITE_TABLE_NAME + "." + SiteEntry.SITE_TABLE_COLUMN_REEF_ID + "=" + Integer.toString(
selectedReef.getReefId() ), null, null);

            case ID_LOAD_RHIS_TABLE:

                cursorsStillLoadingData.put( ID_LOAD_RHIS_TABLE, 1 );

                return new CursorLoader(this, RhisEntry.CONTENT_URI, null, null, null, null);

            default:

                throw new RuntimeException("Loader Not Implemented: " + loaderId);

        }

    }

//
//
// onLoadFinished function
//
//
    @Override
    public void onLoadFinished( Loader<Cursor> loader, Cursor data ) {

        int loaderId = loader.getId();

        switch (loaderId) {

            case ID_LOAD_REEF_TABLE:

                // Clear the old list data
                reefList.clear();

                // Load data from the database voyageTable
                while (data.moveToNext()) {
                    reefList.add( new Reef( data ) );
                }

                break;
```

```
case ID_LOAD_REEFPOLYGONS_TABLE:

    // Here, we know that the cursor is returning only ReefPolygonPoints from this
    // particular Reef, so we can just load them all into the appropriate
    // ReefPolygon

    // Clear the old list data
    reefPolygonPointsList.clear();

    while (data.moveToNext()) {
        reefPolygonPointsList.add( new ReefPolygonPoint( data ) );
    }

    if ( !reefPolygonPointsList.isEmpty() ) {

        reefPolygon = new ReefPolygon( reefPolygonPointsList );

    }

    // Display Reef Polygon
    displayMapFragment.DisplayReefOutline( selectedReef, reefPolygon );

    // Next, we load the details of the Voyages that have visited this Reef
    getSupportLoaderManager().restartLoader(ID_LOAD_VOYAGE_TABLE, null, this);

    break;

case ID_LOAD_SITE_TABLE:

    // Clear the old list data
    siteList.clear();

    // Load data from the database voyageTable
    while (data.moveToNext()) {
        siteList.add( new Site( data ) );
    }

    getSupportLoaderManager().restartLoader(ID_LOAD_MANTA_TABLE, null, this);

    break;

case ID_LOAD_SITEPOLYGONS_TABLE:

    // Here, we know that the query will have returned polygon points from every
    // Site at the Reef, and that the polygon points will be in contiguous chunks for
    // each Site, and in their correct order. Therefore, we can cycle through the list,
    // adding SitePolygonPoints to a list until the __siteId changes, then save the list
    // to a new SitePolygon as necessary
```

```
                // Clear the old lists data
                sitePolygonPointsList.clear();
                sitePolygonsList.clear();

                SitePolygonPointList sitePolygonPointList = new SitePolygonPointList();

                int currentSiteId = -1;

                while ( data.moveToNext() ) {

                    if ( currentSiteId == data.getInt(data.getColumnIndex(
SitePolygonsEntry.SITE_POLYGONS_TABLE_COLUMN_SITE_ID ) ) ){

                        sitePolygonPointList.add( new SitePolygonPoint( data ) );

                    } else {

                        if ( currentSiteId != -1 ) {

                                sitePolygonsList.add( new SitePolygon( sitePolygonPointList ) );
                        }

                        sitePolygonPointList = new SitePolygonPointList();

                        currentSiteId = data.getInt(data.getColumnIndex(
SitePolygonsEntry.SITE_POLYGONS_TABLE_COLUMN_SITE_ID ) );

                    }

                }

                displayMapFragment.addDivesToMap( siteList, diveList, sitePolygonsList );

                break;

            case ID_LOAD_VESSEL_TABLE:

                // Clear the old list data
                vesselList.clear();

                // Load data from the database voyageTable
                while (data.moveToNext()) {
                    vesselList.add( new Vessel( data ) );
                }

                break;

            case ID_LOAD_VOYAGE_TABLE:

                // Clear the old list data
                voyageList.clear();
```

```
            // Load data from the database voyageTable
            while (data.moveToNext()) {

                voyageList.add( new Voyage( data ) );

            }

            getSupportLoaderManager().restartLoader(ID_LOAD_SITE_TABLE, null, this);

            break;

        case ID_LOAD_DIVE_TABLE:

            cursorsStillLoadingData.put( ID_LOAD_DIVE_TABLE, 1 );

            // Clear the old list data
            diveList.clear();

            while (data.moveToNext()) {

                diveList.add( new Dive( data ) );

            }

            displayInfoFragment.DisplayReefInfo( this, mainMap, selectedReef, reefList, siteList,
diveList, mantaList, true );

            getSupportLoaderManager().restartLoader(ID_LOAD_SITEPOLYGONS_TABLE, null, this);

            break;

        case ID_LOAD_MANTA_TABLE:

            // Clear the old list data
            mantaList.clear();

            // Load data from the database voyageTable
            while (data.moveToNext()) {

                mantaList.add( new Manta( data ) );

            }

            displayMapFragment.addMantasToMap( mantaList );

            getSupportLoaderManager().restartLoader(ID_LOAD_DIVE_TABLE, null, this);

            break;

        case ID_LOAD_RHIS_TABLE:
```

```
                // Clear the old list data
                rhisList.clear();

                // Load data from the database voyageTable
                while (data.moveToNext()) {
                    rhisList.add( new Rhis( data ) );
                }

                break;

            default:

                throw new RuntimeException("Loader Not Implemented: " + loaderId);

        }

        displayMap();

    }

    private void displayMap(){

        switch ( mapDisplayType ) {

            case MAP_DISPLAY_ALL_REEFS_WITH_VOYAGES:

                displayMapFragment.DisplayAllReefsControlled( reefList, true);

                break;

            case MAP_DISPLAY_REEF_WITH_SITES:

                findViewById( R.id.display_info_fragment_container_view ).setVisibility( View.VISIBLE );

                displayMapFragment.DisplayReef( selectedReef, reefPolygon );

                break;

            default:

                throw new UnsupportedOperationException( "That mapDisplayType not implemented yet." );

        }

    }

    public void LoadNewData() {

        LoadNewData.loadNewData( this, reefList, mantaList );
```

```
        getSupportLoaderManager().restartLoader(ID_LOAD_SITE_TABLE, null, this);
        getSupportLoaderManager().restartLoader(ID_LOAD_DIVE_TABLE, null, this);
        getSupportLoaderManager().restartLoader(ID_LOAD_MANTA_TABLE, null, this);
        getSupportLoaderManager().restartLoader(ID_LOAD_VOYAGE_TABLE, null, this);

        // Ask map to turn off previous Mantas and Culls

        // Ask map to display new Mantas

    }

    @Override
    public void onLoaderReset(Loader<Cursor> loader) {

    }

    private void checkPermission() {
        if (ContextCompat.checkSelfPermission(this,
                Manifest.permission.WRITE_EXTERNAL_STORAGE)
                != PackageManager.PERMISSION_GRANTED && ContextCompat.checkSelfPermission(this,
                Manifest.permission.READ_EXTERNAL_STORAGE)
                != PackageManager.PERMISSION_GRANTED) {//Can add more as per requirement

            ActivityCompat.requestPermissions(this,
                    new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE,Manifest.permission.READ_EXTERNAL_STORAGE},
                    123);

        } else {

        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           String permissions[], int[] grantResults)
    {
        switch (requestCode) {
            case 123: {


                if (grantResults.length > 0
                        && grantResults[0] == PackageManager.PERMISSION_GRANTED)     {
                    //Peform your task here if any
                } else {

                    checkPermission();
                }
                return;
            }
```

```
        }
    }

    @Override
    public void sendTouchedReefId(int reefId) {

        selectedReef = reefList.get( 0 );

        for ( Reef reef : reefList ){

            if ( reef.getReefId() == reefId ){

                selectedReef = reef;

            }

        }

        mapDisplayType = MAP_DISPLAY_REEF_WITH_SITES;

        // First, we load the reefPolygon, and then display it
        getSupportLoaderManager().restartLoader(ID_LOAD_REEFPOLYGONS_TABLE, null, this);

    }

    @Override
    public void sendMapTouched() {

        findViewById( R.id.display_info_fragment_container_view ).setVisibility( View.GONE );

    }

    @Override
    public void sendGenerateWorkplanButtonPress() {

        ImplementDecisionTreeAtReef implementDecisionTreeAtReef = new ImplementDecisionTreeAtReef(
selectedReef.getReefName(), siteList, diveList, mantaList );
        String workplanText = implementDecisionTreeAtReef.ImplementDecisionTreeAtReefAndFindControlTasks();

        displayInfoFragment.DisplayWorkplanInfo( workplanText );

        List<Integer> siteIdsInControlOrder =
implementDecisionTreeAtReef.ImplementDecisionTreeAtReefAndFindSiteIdsToBeControlledInOrder();

        if ( !siteIdsInControlOrder.isEmpty() ) {

            displayMapFragment.DisplayWorkplanMarkers(this, mainMap, selectedReef, reefPolygon, reefList,
siteList, diveList, sitePolygonsList, mantaList, true, siteIdsInControlOrder);

        }
```

```
    }



    @Override
    public void sendLoadDataButtonPress() {

        LoadNewData();

    }

    @Override
    public void sendMapFragment( DisplayMap displayMap ) {

        displayMapFragment = displayMap;

    }

    @Override
    public void sendInfoFragment( DisplayInfo displayInfo ) {

        displayInfoFragment = displayInfo;

    }

}
```

## B.2 DisplayMap.java

```
// COMMENT OUT FOR OFFLINE USE
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0;

import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Color;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.ViewTreeObserver;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;

import androidx.core.graphics.ColorUtils;
import androidx.fragment.app.Fragment;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
```

```
import com.google.android.gms.maps.GoogleMap.OnMarkerClickListener;
import com.google.android.gms.maps.GoogleMap.OnMapClickListener;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.LatLngBounds.Builder;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;
import com.google.android.gms.maps.model.Polygon;
import com.google.android.gms.maps.model.PolygonOptions;
import com.google.android.gms.maps.model.Polyline;
import com.google.android.gms.maps.model.PolylineOptions;
import com.google.maps.android.clustering.ClusterManager;

import java.util.ArrayList;
import java.util.List;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Manta;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.MantaList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Reef;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.DiveList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.ReefPolygon;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Site;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.SiteList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.SitePolygon;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.SitePolygonList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Voyage;

import static java.lang.Math.max;

// The minimum information required to create a new DisplayMap fragment object is nothing - the map
// simply displays at the default Google location. After that, various simple helper methods are
// provided to move the map to a certain location, to zoom the map etc.
//
// Custom methods are then provided to interact specifically with COTS Program data, as follows:
//
// 1) DisplayReefs( ReefList reefList ): Zoom to the extent of Reefs in the ReefList and add a marker at the LatLng of each
Reef
// 2) DisplayReef( Reef reef ): Zoom to the Lat Lng of the Reef with Zoom level 7.0f
// 3) DisplayReef( Reef reef, SiteList siteList ): Add markers at the LatLng of each Site at the Reef
// 4) DisplayMantas( MantaList mantaList ): Add Polylines to the current map for the mantaList
// 5) DisplayDives( DiveList diveList, SitePolygonList sitePolygonList ): Add Site Polygons
// is provided

public class DisplayMap extends Fragment implements
        OnMapReadyCallback,
        OnMarkerClickListener,
        OnMapClickListener,
        View.OnClickListener {

    private ClusterManager<MyItem> clusterManager;

    private List<Marker> mapMarkerList = new ArrayList<>();
    private List<Polygon> mapPolygonList = new ArrayList<>();
    private List<Polyline> mapPolylineList = new ArrayList<>();
```

```
private List<Marker> workplanMarkerList = new ArrayList<>();

private Button mantaButton;
private Button cullButton;

private boolean mantaButtonClicked = false;
private boolean cullButtonClicked = false;

private int mapDisplayType;
private final int MAP_DISPLAY_ALL_REEFS_WITH_VOYAGES = 100;
private final int MAP_DISPLAY_REEF_WITH_SITES = 200;

private GoogleMap mainMap;
private View mapView;
private View mapFrameView;

private int mapWidth = 0;
private int mapHeight = 0;

// Define ArrayLists of the fundamental data types associated with the Control Program data
public List<Reef> dmReefList = new ArrayList<Reef>();

private boolean mapDataReady = false;

private Context dmContext;

// Latitude and longitude of Cairns
private double CAIRNS_LATITUDE = -16.9186;
private double CAIRNS_LONGITUDE = 145.7781;
private LatLng cairnsLatLng = new LatLng( CAIRNS_LATITUDE, CAIRNS_LONGITUDE );

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent, Bundle savedInstanceState) {

    // Define the xml file for the fragment
    mapView = inflater.inflate(R.layout.display_map, parent, false);

    return mapView;

}

// This event is triggered soon after onCreateView().
// Any view setup should occur here.  E.g., view lookups and attaching view listeners.
@Override
public void onViewCreated( final View view, Bundle savedInstanceState ) {

    // Define the view for the actual map fragment
    mapFrameView = mapView.findViewById( R.id.map_frame );

    // Set up the map fragment
    ( ( SupportMapFragment ) getChildFragmentManager().findFragmentById( R.id.map_main ) ).getMapAsync( this );

    // Because the view gets resized dynamically as the fragment and other fragments are added
    // and removed from the screen, we need to add a listener to update the current width and
    // height available to draw the map.
```

```
        view.getViewTreeObserver().addOnGlobalLayoutListener(

                new ViewTreeObserver.OnGlobalLayoutListener() {

                    @Override
                    public void onGlobalLayout() {

//                        mapFrameView.getViewTreeObserver().removeOnGlobalLayoutListener(this);

                        mapWidth = mapFrameView.getHeight();
                        mapHeight = mapFrameView.getHeight();

                    }

                });

        // Set up handles
        mantaButton = mapView.findViewById(R.id.mantaButton);
        cullButton = mapView.findViewById(R.id.cullButton);

        mantaButton.setOnClickListener( this );
        cullButton.setOnClickListener( this );

        displayMapFragmentReadyListener.sendMapFragment( this );

    }

    //    Fragment constructor
    public DisplayMap() {

        super();

    }

    private void DisplayCairnsRegion() {

        mainMap.clear();

        mainMap.moveCamera( CameraUpdateFactory.newLatLngZoom( cairnsLatLng, 7.0f) );

    }

    public void DisplayAllVoyages(Context context, final GoogleMap map, List<Voyage> voyageList, boolean firstZoom ) {

        dmContext = context;

        mainMap.clear();
        if ( clusterManager != null ) {

            clusterManager.clearItems();

        } else {

            clusterManager = new ClusterManager<MyItem>( context, map );
```

```
        }

        mainMap.setOnCameraIdleListener(clusterManager);

        Builder builder = new Builder();

        for (Voyage voyage : voyageList) {

            LatLng latLng = new LatLng(voyage.meanLatitude(), voyage.meanLongitude());

            builder.include(latLng);

            clusterManager.addItem(new MyItem(latLng.latitude, latLng.longitude, "Barry",
String.valueOf(voyage.getVoyageId())));

        }

        if (firstZoom) {

            mainMap.moveCamera(CameraUpdateFactory.newLatLngBounds(builder.build(), mapWidth, mapHeight, 200));

        } else {

            mainMap.animateCamera(CameraUpdateFactory.newLatLngBounds(builder.build(), mapWidth, mapHeight, 200));

        }

    }

    public void DisplayAllReefsControlled( List<Reef> reefList, boolean firstZoom ) {

        mainMap.clear();
        mapMarkerList.clear();
        mapPolygonList.clear();
        mapPolylineList.clear();

        // If we're displaying all reefs, we don't want to display the display button bar
        mapView.findViewById(R.id.mantaButton).setVisibility(View.GONE);
        mapView.findViewById(R.id.cullButton).setVisibility(View.GONE);

        // Update DisplayMap's dmReefList
        dmReefList = reefList;
        mapDataReady = true;

        // Set up a new cluster manager if necessary, or clear the old one
        // Also set up the required OnCameraIdle listener so that the ClusterManager can do it's thing
        // when we stop scrolling
        if ( clusterManager != null ) {

            clusterManager.clearItems();

        } else {

            clusterManager = new ClusterManager<MyItem>( this.getContext(), mainMap );
```

```
        }

        mainMap.setOnCameraIdleListener( clusterManager );

        // Set up a builder to store the lats and longs of all the reef polygon points, then
        // cycle through add all the points for the Reef Polygon to the builder
        Builder builder = new Builder();

        for ( Reef reef : reefList ) {

            LatLng latLng = new LatLng(reef.getReefLatitude(), reef.getReefLongitude());

            builder.include(latLng);

            clusterManager.addItem(new MyItem(latLng.latitude, latLng.longitude, String.valueOf(reef.getReefName()),
String.valueOf(reef.getReefId())));

        }

        // Zoom the map to the appropriate location. If it's the first time the map is set up, we
        // just move the camera straight to the location - otherwise we animate the move
        if ( firstZoom ) {

            mainMap.moveCamera( CameraUpdateFactory.newLatLngBounds( builder.build(), mapWidth, mapHeight, 200 ) );

        } else {

            mainMap.animateCamera( CameraUpdateFactory.newLatLngBounds( builder.build(), mapWidth, mapHeight, 200 ) );

        }

    }

    public void DisplayReefOutline( Reef reef, ReefPolygon reefPolygon ) {

        // Clear the map and clear all the clusters
        mainMap.clear();
        mapMarkerList.clear();
        mapPolygonList.clear();
        mapPolylineList.clear();

        if ( clusterManager != null ) {

            clusterManager.clearItems();

        }

        // Set up a builder to store the lats and longs of all the reef polygon points, then
        // cycle through add all the points for the Reef Polygon to the builder. At a minimum
        // add the lat lng recorded for the Reef, in case there are no polygon points
        Builder builder = new Builder();

        builder.include( new LatLng( reef.getReefLatitude(), reef.getReefLongitude() ) );

        if ( !( reefPolygon == null ) ) {
```

```
            mainMap.addPolygon(new
PolygonOptions().addAll(reefPolygon.getReefPolygonPoints()).strokeWidth(5).strokeColor(Color.GRAY));

            for (LatLng reefPolygonPoint : reefPolygon.getReefPolygonPoints()) {

                builder.include(reefPolygonPoint);

            }
        }

        // Zoom camera early to provide quick UI feedback
        mainMap.animateCamera( CameraUpdateFactory.newLatLngBounds( builder.build(), mapWidth, mapHeight, 200 ) );

    }


    public void DisplayReef( final Reef reef, ReefPolygon reefPolygon ) {

        // Set up a builder to store the lats and longs of all the reef polygon points, then
        // cycle through add all the points for the Reef Polygon to the builder. At a minimum
        // add the lat lng recorded for the Reef, in case there are no polygon points
        Builder builder = new Builder();

        builder.include( new LatLng( reef.getReefLatitude(), reef.getReefLongitude() ) );

        if ( !( reefPolygon == null ) ) {

            for ( LatLng reefPolygonPoint : reefPolygon.getReefPolygonPoints() ) {

                builder.include(reefPolygonPoint);

            }

        }

        // Zoom camera early to provide quick UI feedback
        mainMap.animateCamera( CameraUpdateFactory.newLatLngBounds( builder.build(), mapWidth, mapHeight, 40 ) );

        mapView.findViewById( R.id.loadingPanel).setVisibility(View.GONE);

    }

    public void addDivesToMap( final SiteList siteList, final DiveList diveList, final SitePolygonList sitePolygonsList ){

        // First, find the most recent Dive in the Sites at the Reef
        Integer diveMostRecentVoyageId = diveList.getMostRecentDiveVoyageId();

        DiveList diveListOnMostRecentDiveVoyage = diveList.getDivesByVoyageId( diveMostRecentVoyageId );

        // We know that the function has been passed data about the current Reef, so all the Sites
        // in siteList are at the Reef, and all the Sites at the Reef are in siteList. In future,
        // if this is not the case, we might need to select Sites at the Reef.
        // for ( Site site : siteList.getSitesWithReefId( reefId ) ) {
        for ( Site site : siteList ) {
```

```
            SitePolygon siteSitePolygon = sitePolygonsList.getSitePolygonBySiteId(site.getSiteId());


            if ( siteSitePolygon != null ) {


                PolygonOptions cullPolygonOptions;


                Integer totalCOTSCulledOnDivesAtSiteOnMostRecentVoyageToReef =
diveListOnMostRecentDiveVoyage.getDivesBySiteId(site.getSiteId()).getTotalCOTS();


                // THIS SEARCHES FOR THE MOST RECENT DIVE AND MANTA TWICE - WHY?

//                Integer totalCOTSCulledOnDivesAtSiteOnMostRecentVoyageToReef =
diveListOnMostRecentDiveVoyage.getDivesBySiteId(site.getSiteId()).getDivesOnMostRecentVoyage().getTotalCOTS();


                // Add the polygon to the map, coloured by the number of COTS removed during the last cull there - if there
were no COTS culled there, don't provide any fill

                cullPolygonOptions = new
PolygonOptions().addAll(siteSitePolygon.getSitePolygonPoints()).strokeWidth(1).fillColor(colorFunctionCull(totalCOTSCulledOnD
ivesAtSiteOnMostRecentVoyageToReef, 64));


                mapPolygonList.add( mainMap.addPolygon( cullPolygonOptions ) );


            }


        }


        cullButtonClicked = false;


        cullButton.setVisibility(View.VISIBLE);


    }


    public void addMantasToMap( final MantaList mantaList ){


        // Find the most recent Mantas at the Reef

        Integer mantaMostRecentVoyageId = mantaList.getMostRecentMantaVoyageId();


        MantaList mantaListOnMostRecentMantaVoyage = mantaList.getMantasByVoyageId( mantaMostRecentVoyageId );


        // Add all the mantas appropriated coloured

        for (Manta manta : mantaListOnMostRecentMantaVoyage) {


            PolylineOptions mantaPolylineOptions = new PolylineOptions().add(new LatLng(manta.getMantaStartLat(),
manta.getMantaStartLong()), new LatLng(manta.getMantaStopLat(),
manta.getMantaStopLong())).width(3).color(colorFunctionManta(mantaCOTSSeverity(manta), 255));


            mapPolylineList.add(mainMap.addPolyline(mantaPolylineOptions));


        }


        mantaButtonClicked = false;


        mantaButton.setVisibility(View.VISIBLE);




    }
```

```
    private int mantaCOTSSeverity( Manta manta ) {

        int mantaCOTSseverity = 0;

        if ( manta.getMantaScars().equals( "c" ) || ( manta.getMantaScars().equals( "p" ) && ( manta.getMantaCOTS() > 1 ) ) )
{

            mantaCOTSseverity = max( 2, mantaCOTSseverity );

        } else if ( manta.getMantaScars().equals( "p" ) || ( manta.getMantaScars().equals( "a" ) && ( manta.getMantaCOTS() >
0 ) ) ) {

            mantaCOTSseverity = 1;

        } else {

            mantaCOTSseverity = 0;

        }

        return mantaCOTSseverity;

    }

    private int colorFunctionCull(Integer cotsCount, int opacity) {

        int outputColor;

        if (cotsCount == null ){
            outputColor = ColorUtils.setAlphaComponent(Color.WHITE, 00);
        } else if (cotsCount < 1) {
            outputColor = ColorUtils.setAlphaComponent(Color.GREEN, opacity);
        } else if (cotsCount < 4) {
            outputColor = ColorUtils.setAlphaComponent(Color.YELLOW, opacity);
        } else {
            outputColor = ColorUtils.setAlphaComponent(Color.RED, opacity);
        }

        return outputColor;

    }

    private int colorFunctionManta(int cotsCount, int opacity) {

        int outputColor;

        if (cotsCount < 1) {
            outputColor = ColorUtils.setAlphaComponent(Color.GREEN, opacity);
        } else if (cotsCount < 2) {
            outputColor = ColorUtils.setAlphaComponent(Color.YELLOW, opacity);
        } else {
            outputColor = ColorUtils.setAlphaComponent(Color.RED, opacity);
        }

        return outputColor;
```

```
    }

    public void DisplayWorkplanMarkers( Context context, final GoogleMap map, final Reef reef, ReefPolygon reefPolygon, final
List<Reef> reefList, final SiteList siteList, final DiveList diveList, final SitePolygonList sitePolygonsList, final
MantaList mantaList, Boolean zoom, List<Integer> siteIdsInControlOrder ){

        int i = 1;

        for (Integer siteId : siteIdsInControlOrder) {

            Site site = siteList.getSiteBySiteId(siteId);

            LinearLayout tv = ((LinearLayout) ((Activity) context).getLayoutInflater().inflate(R.layout.site_marker_info,
null, false));
            TextView site_marker_info_text = tv.findViewById(R.id.site_marker_info_text);

            //            site_marker_info_text.setText( Integer.toString( i ) + ", " + site.getSiteName() + ", COTS: " +
numberOfCOTSCulledAtSite +", Manta: " + numberOfMantaCOTSAtSite );
            site_marker_info_text.setText(Integer.toString(i));

            tv.measure(View.MeasureSpec.makeMeasureSpec(0, View.MeasureSpec.UNSPECIFIED),
                    View.MeasureSpec.makeMeasureSpec(0, View.MeasureSpec.UNSPECIFIED));
            tv.layout(0, 0, tv.getMeasuredWidth(), tv.getMeasuredHeight());

            tv.setDrawingCacheEnabled(true);
            tv.buildDrawingCache();
            Bitmap bm = tv.getDrawingCache();

            workplanMarkerList.add(

                    mainMap.addMarker(

                            new MarkerOptions()
                                    .position(new LatLng(site.getSiteLatitude(), site.getSiteLongitude()))
                                    .title(Integer.toString(i))
                                    .icon(BitmapDescriptorFactory.fromBitmap(bm))

                    )

            );

            i++;

        }


    }


    @Override
    public boolean onMarkerClick( final Marker marker ) {

        mapView.findViewById( R.id.loadingPanel ).setVisibility( View.VISIBLE );

        if ( marker.getSnippet() != null ) {

            int id = Integer.parseInt(marker.getSnippet());
```

```
        sendTouchedReefId( id );

        //TODO: Stagger loading to make UI responsive - first search and load dives and mantas
        // just from the most recent Voyage and display quickly, then move to background loading
        // all the dives and mantas at the relevant Reef so that if the user requires it, the
        // data is ready to go

    }

    return true;
}


@Override
public void onMapClick( final LatLng latlng ) {

    if ( mapDataReady ) {

        mapDisplayType = MAP_DISPLAY_ALL_REEFS_WITH_VOYAGES;

        DisplayAllReefsControlled( dmReefList, false );

        sendMapTouched();

    } else {

        // Do nothing

    }


}

@Override
public void onMapReady( final GoogleMap map ) {

    mainMap = map;

    mainMap.setOnMarkerClickListener( this );

    mainMap.setOnMapClickListener( this );

    DisplayCairnsRegion();

}

private DisplayMapFragmentMarkerTouchListener displayMapFragmentMarkerTouchListener;
private DisplayMapFragmentMapTouchListener displayMapFragmentMapTouchListener;
private DisplayMapFragmentReadyListener displayMapFragmentReadyListener;

private void sendTouchedReefId( int reefId ) {

    if  ( displayMapFragmentMarkerTouchListener != null ) {
```

```
            displayMapFragmentMarkerTouchListener.sendTouchedReefId( reefId );
        }
    }
    private void sendMapTouched() {
        if  ( displayMapFragmentMapTouchListener != null ) {
            displayMapFragmentMapTouchListener.sendMapTouched();
        }
    }
    private void sendMapFragment( DisplayMap displayMap ){
        if  ( displayMapFragmentReadyListener != null ) {
            displayMapFragmentReadyListener.sendMapFragment( displayMap );
        }
    }
    public interface DisplayMapFragmentMarkerTouchListener {
        public void sendTouchedReefId( int reefId );
    }
    public interface DisplayMapFragmentMapTouchListener {
        public void sendMapTouched();
    }
    public interface DisplayMapFragmentReadyListener {
        public void sendMapFragment( DisplayMap displayMap );
    }

    @Override
    public void onAttach( Context context ) {
        super.onAttach( context );
        try {
            displayMapFragmentMarkerTouchListener = ( DisplayMapFragmentMarkerTouchListener ) context;
            displayMapFragmentMapTouchListener = ( DisplayMapFragmentMapTouchListener ) context;
```

```
        displayMapFragmentReadyListener = ( DisplayMapFragmentReadyListener ) context;

    } catch ( ClassCastException e ) {

        throw new ClassCastException( context.toString()+ " must implement displayMapFragmentMarkerTouchListener" );

    }

}

@Override
public void onDetach() {

    displayMapFragmentMarkerTouchListener = null;

    displayMapFragmentMapTouchListener = null;

    super.onDetach();

}

private void toggleMantas(){

    mantaButtonClicked = !mantaButtonClicked;

    for ( Polyline mantaPolyline : mapPolylineList ) {

        mantaPolyline.setVisible( !mantaButtonClicked );

    }

}

private void toggleDives(){

    cullButtonClicked = !cullButtonClicked;

    int alpha;

    if (cullButtonClicked) {
        alpha = 00;
    } else {
        alpha = 64;
    }

    for (Polygon cullPolygon : mapPolygonList) {

        cullPolygon.setFillColor(ColorUtils.setAlphaComponent(cullPolygon.getFillColor(), alpha));

    }

}

@Override
```

```
    public void onClick( View view ) {

        switch ( view.getId() ) {

            case R.id.mantaButton:

                toggleMantas();

                break;

            case R.id.cullButton:

                toggleDives();

                break;

            default:

        }

    }


}
```

## B.3 DisplayInfo.java

```
// COMMENT OUT FOR OFFLINE USE
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;

import androidx.fragment.app.Fragment;

import com.google.android.gms.maps.GoogleMap;

import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.concurrent.TimeUnit;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.DiveList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.MantaList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Reef;
```

```
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Site;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.SiteList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Voyage;

import static java.lang.Math.max;

/**
 * Created by fle125 on 14/04/2017.
 */

public class DisplayInfo extends Fragment implements
        View.OnClickListener {

    private View infoView;

    private View infoPanelView;
    private View workplanPanelView;

    private TextView reefInfoOverlayReefName;
    private TextView reefInfoOverlayReefMode;
    private TextView reefInfoOverlayNumberOfSites;
    private TextView reefInfoOverlayLastCullDate;
    private TextView reefInfoOverlayLastMantaDate;
    private TextView reefInfoOverlayNumberOfDaysSinceLastManta;
    private TextView reefInfoOverlayMantaDue;
    private TextView reefInfoOverlayTotalCOTSCulledDuringLastCull;
    private TextView reefInfoOverlayTotalCOTSSeenDuringLastManta;
    private TextView reefInfoOverlayAnyMantaScarsSeenDuringLastManta;
    private TextView reefInfoOverlayNumberOfSitesWithMantaCOTSOrScars;

    private TextView reefInfoOverlayWorkPlan;

    private Button generateWorkPlanButton;
    private Button loadSurveillanceFileButton;

    @Override
    public View onCreateView( LayoutInflater inflater, ViewGroup parent, Bundle savedInstanceState ) {

        // Define the xml file for the fragment
        infoView = inflater.inflate( R.layout.display_info, parent, false );

        return infoView;
    }

    // This event is triggered soon after onCreateView().
    // Any view setup should occur here.  E.g., view lookups and attaching view listeners.
    @Override
    public void onViewCreated( View view, Bundle savedInstanceState ) {

        // Setup handles to view objects
```

```
        workplanPanelView = infoView.findViewById( R.id.overlay_workplan_panel );
        infoPanelView = infoView.findViewById( R.id.overlay_info_panel );

        reefInfoOverlayReefName = infoView.findViewById( R.id.reef_info_overlay_reef_name );
        reefInfoOverlayReefMode = infoView.findViewById( R.id.reef_info_overlay_reef_reef_mode );
        reefInfoOverlayNumberOfSites = infoView.findViewById( R.id.reef_info_overlay_reef_number_of_sites
);
        reefInfoOverlayLastCullDate = infoView.findViewById( R.id.reef_info_overlay_last_cull_date );
        reefInfoOverlayLastMantaDate = infoView.findViewById( R.id.reef_info_overlay_last_manta_date );
        reefInfoOverlayNumberOfDaysSinceLastManta = infoView.findViewById(
R.id.reef_info_overlay_number_of_days_since_last_manta );
        reefInfoOverlayMantaDue = infoView.findViewById( R.id.reef_info_overlay_manta_due );
        reefInfoOverlayTotalCOTSCulledDuringLastCull = infoView.findViewById(
R.id.reef_info_overlay_reef_total_cots_culled_during_last_cull );
        reefInfoOverlayTotalCOTSSeenDuringLastManta = infoView.findViewById(
R.id.reef_info_overlay_reef_total_cots_seen_during_last_manta );
        reefInfoOverlayAnyMantaScarsSeenDuringLastManta = infoView.findViewById(
R.id.reef_info_overlay_reef_any_manta_scars_seen_during_last_manta );
        reefInfoOverlayNumberOfSitesWithMantaCOTSOrScars = infoView.findViewById(
R.id.reef_info_overlay_reef_number_of_sites_with_manta_cots_or_scars );

        reefInfoOverlayWorkPlan = infoView.findViewById(R.id.reef_info_overlay_workplan);

        generateWorkPlanButton = infoView.findViewById(R.id.generateWorkplanButton);
        loadSurveillanceFileButton = infoView.findViewById(R.id.loadSurveillanceFileButton);

        generateWorkPlanButton.setOnClickListener( this );
        loadSurveillanceFileButton.setOnClickListener( this );

        displayInfoFragmentReadyListener.sendInfoFragment( this );

    }

    //    Fragment constructor
    public DisplayInfo() {

        super();

    }

    // In future, we could decide to populate a text info view for when all Voyages are displayed on
    // the map, but we don't use it for now
    public static void DisplayAllVoyageInfo(Context context, final GoogleMap map, List<Voyage> voyageList,
boolean firstZoom) {

    }

    // In future, we could decide to populate a text info view for when all Voyages with control
    // data are displayed on the map, but we don't use it for now
    public static void DisplayAllReefsControlled(Context context, final GoogleMap map, List<Reef> reefList,
boolean firstZoom) {

    }
```

```
    // Once a Reef is selected, we want to display a range of information about it in TextViews.
    // However, this is not as trivial as it might at first seem, because some of the information
    // we want to display as a simple statement of, for instance, number of days since the Reef was
    // last Manta towed, needs to be generated by analysing various underlying data.
    //
    // That means we have to load the underlying data into this method, and process it
    // appropriately. If the calculation relates specifically to one of our custom Types,
    // calculating the total number of COTS culled during a Dive, for instance, then we
    // try to house it as a method of the Type itself. If it relates specifically to a list of one
    // of our Types, for instance finding the most recent Dive at Sites in a SiteList, we try to
    // house it in the custom TypeList. If it is neither of these, we house it as a private method
    // of this class.
    public void DisplayReefInfo( final Context context, final GoogleMap map, final Reef reef, final
List<Reef> reefList, final SiteList siteList, final DiveList diveList, final MantaList mantaList, Boolean
zoom) {


        MainActivity.loaderTiming.addSplit("DisplayReefInfoStart");


        int totalCOTSCount = 0;
        int totalMantaCount = 0;
        boolean anyMantaScars = false;
        int numberOfSitesWithMantaCOTSOrScars = 0;


        Date mostRecentMantaDate = new Date(0, 01, 01); /* Set latest Date to a Date before the program
began  */
        Date mostRecentDiveDate = new Date(0, 01, 01); /* Set latest Date to a Date before the program
began */


        Integer diveMostRecentVoyageId = diveList.getMostRecentDiveVoyageId();
        Integer mantaMostRecentVoyageId = mantaList.getMostRecentMantaVoyageId();


        DiveList diveListOnMostRecentDiveVoyage = diveList.getDivesByVoyageId( diveMostRecentVoyageId );
        MantaList mantaListOnMostRecentMantaVoyage = mantaList.getMantasByVoyageId( mantaMostRecentVoyageId
);


        // Then, find the date of the most recent Dive and Manta in the Sites at the Reef
        mostRecentDiveDate = diveListOnMostRecentDiveVoyage.get(0).getDiveDateAsDate();
        mostRecentMantaDate = mantaListOnMostRecentMantaVoyage.get(0).getMantaDateAsDate();


        // We know that the function has been passed data about the current Reef, so all the Sites
        // in siteList are at the Reef, and all the Sites at the Reef are in siteList. In future,
        // if this is not the case, we might need to select Sites at the Reef.
        for ( Site site : siteList ) {


            // We total up the number of COTS culled and detected in mantas for each
            totalCOTSCount += diveListOnMostRecentDiveVoyage.getDivesBySiteId( site.getSiteId()
).getTotalCOTS();


            totalMantaCount += mantaListOnMostRecentMantaVoyage.getMantasBySiteId( site.getSiteId()
).getTotalCOTS();


// THIS SEARCHES FOR THE MOST RECENT DIVE AND MANTA TWICE - WHY?
//            totalCOTSCount += diveListOnMostRecentDiveVoyage.getDivesBySiteId( site.getSiteId()
).getDivesOnMostRecentVoyage().getTotalCOTS();
```

```
//
//            totalMantaCount += mantaListOnMostRecentMantaVoyage.getMantasBySiteId( site.getSiteId()
).getMantasOnMostRecentVoyage().getTotalCOTS();




        }


        boolean reefModeQ = ( totalMantaCount == 0 ) || ( totalCOTSCount == 0 );


        Calendar todaysDate = Calendar.getInstance();
        long timeSinceLastManta = todaysDate.getTime().getTime() - mostRecentMantaDate.getTime();
        long numberOfDaysSinceLastManta = TimeUnit.MILLISECONDS.toDays(timeSinceLastManta);
        int mantaThreshold;
        if ( reefModeQ ) {
            mantaThreshold = MainActivity.daysToMantaTowIntensiveControlReef;
        } else {
            mantaThreshold = MainActivity.daysToMantaTowMaintenanceReef;
        }
        boolean mantaDue = (numberOfDaysSinceLastManta > mantaThreshold);


        SimpleDateFormat simpleDate = new SimpleDateFormat("dd/MM/yyyy");


        String reefMode;
        if ( reefModeQ ) {
            reefMode = "Maintenance Mode";
        } else {
            reefMode = "Intensive Mode";
        }


        reefInfoOverlayReefName.setText( reef.getReefName() );
        reefInfoOverlayReefMode.setText( "Reef Mode (calculated): " + reefMode );
        reefInfoOverlayNumberOfSites.setText( "Number of Sites at Reef: " + Integer.toString(
siteList.size() ) );
        reefInfoOverlayLastCullDate.setText( "Last cull date: " + simpleDate.format( mostRecentDiveDate )
);
        reefInfoOverlayLastMantaDate.setText( "Last manta tow date: " + simpleDate.format(
mostRecentMantaDate ) );
        reefInfoOverlayNumberOfDaysSinceLastManta.setText( "Number of days since last manta tow: " +
Long.toString( numberOfDaysSinceLastManta ) );
        reefInfoOverlayMantaDue.setText( "Manta tow due? " + Boolean.toString( mantaDue ) );
        reefInfoOverlayTotalCOTSCulledDuringLastCull.setText( "Total COTS culled during last cull: " +
Integer.toString( totalCOTSCount ) );
        reefInfoOverlayTotalCOTSSeenDuringLastManta.setText( "Total COTS seen during last manta: " +
Integer.toString( totalMantaCount ) );
        reefInfoOverlayAnyMantaScarsSeenDuringLastManta.setText( "Any scars seen during last manta: " +
Boolean.toString( anyMantaScars ) );
        reefInfoOverlayNumberOfSitesWithMantaCOTSOrScars.setText( "Number of Sites with COTS or scars
during manta: " + Integer.toString( numberOfSitesWithMantaCOTSOrScars ) );


        ((Activity) context).findViewById( R.id.reef_info_overlay ).setVisibility( View.VISIBLE );


        (((Activity) context).findViewById(R.id.loadingPanel)).setVisibility(View.GONE);
```

```
    }

    public void DisplayWorkplanInfo( String workplanText ){

        if ( workplanPanelView.getVisibility() == View.VISIBLE ) {

            workplanPanelView.setVisibility(View.GONE);
            infoPanelView.setVisibility(View.VISIBLE);
            // Ask mapFragment to turn off all the markers
            generateWorkPlanButton.setText("Generate Workplan");

        } else {

            workplanPanelView.setVisibility(View.VISIBLE);
            infoPanelView.setVisibility(View.GONE);
            reefInfoOverlayWorkPlan.setText( workplanText );
            // Ask mapFragment to turn off all the markers
            generateWorkPlanButton.setText("Return to info view");

        }

    }



    private DisplayInfoFragmentLoadDataButtonPressListener displayInfoFragmentLoadDataButtonPressListener;
    private DisplayInfoFragmentGenerateWorkplanButtonPressListener
displayInfoFragmentGenerateWorkplanButtonPressListener;
    private DisplayInfoFragmentReadyListener displayInfoFragmentReadyListener;



    private void sendLoadDataButtonPress() {

        if  ( displayInfoFragmentLoadDataButtonPressListener != null ) {

            displayInfoFragmentLoadDataButtonPressListener.sendLoadDataButtonPress();

        }

    }

    private void sendGenerateWorkplanButtonPress() {

        if  ( displayInfoFragmentGenerateWorkplanButtonPressListener != null ) {

            displayInfoFragmentGenerateWorkplanButtonPressListener.sendGenerateWorkplanButtonPress();

        }

    }

    private void sendInfoFragment( DisplayInfo displayInfo ){
```

```
        if  ( displayInfoFragmentReadyListener != null ) {

            displayInfoFragmentReadyListener.sendInfoFragment( displayInfo );

        }

    }

    public interface DisplayInfoFragmentLoadDataButtonPressListener {

        public void sendLoadDataButtonPress();

    }

    public interface DisplayInfoFragmentGenerateWorkplanButtonPressListener {

        public void sendGenerateWorkplanButtonPress();

    }

    public interface DisplayInfoFragmentReadyListener {

        public void sendInfoFragment( DisplayInfo displayInfo );

    }



    @Override
    public void onAttach( Context context ) {

        super.onAttach( context );

        try {

            displayInfoFragmentLoadDataButtonPressListener = (
DisplayInfoFragmentLoadDataButtonPressListener ) context;

            displayInfoFragmentGenerateWorkplanButtonPressListener = (
DisplayInfoFragmentGenerateWorkplanButtonPressListener ) context;

            displayInfoFragmentReadyListener = ( DisplayInfoFragmentReadyListener ) context;



        } catch ( ClassCastException e ) {

            throw new ClassCastException( context.toString()+ " must implement
displayMapFragmentMarkerTouchListener" );

        }
```

```
    }

    @Override
    public void onDetach() {

        displayInfoFragmentLoadDataButtonPressListener = null;

        displayInfoFragmentGenerateWorkplanButtonPressListener = null;

        super.onDetach();

    }


    @Override
    public void onClick( View view ) {

        switch ( view.getId() ) {

            case R.id.loadSurveillanceFileButton:

                sendLoadDataButtonPress();

                break;

            case R.id.generateWorkplanButton:

                sendGenerateWorkplanButtonPress();

                break;

            default:

        }

    }

}
```

## B.4 LoadNewData.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.util.Log;
```

```
import org.json.JSONArray;
import org.json.JSONObject;

import java.io.File;
import java.util.List;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Dive;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.DiveList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Manta;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.MantaList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Reef;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Rhis;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Voyage;

public class LoadNewData {
//TODO: Set this up as a headless fragment so loading is not affected by orientation change

    //    extends Fragment  {

//    @Override
//    public void onCreate( Bundle savedInstanceState ) {
//
//        super.onCreate( savedInstanceState );
//
//        //TODO: Double check if this is what we really want
//        setRetainInstance( true );
//
//    }

    public static void loadNewData( Context context, List<Reef> reefList, MantaList mantaList ) {

        //
        // Android now makes it painfully difficult to just read a folder on an SD card, because it
        // wants to enforce filesystem use to maintain security. This is not useful for us, because
        // we are trying to interact between apps that were developed when you were allowed to read
        // the sdCard directly, and we don't have access to the apps that create the files that are
        // stored there. So, we need a workaround.
        //
        // The workaround is clunky, and depends on several assumptions, which is a non-ideal
        // situation. In the medium-term, we should get ThinkSpatial to recode their apps to use
        // a Uri based file store that the CCC DST app can access.
        //
        // In the short term, we search the items in the directory /storage for the folder with a
        // name of the form ####-####, where the # represent numerals.
        //
        // TODO: Coordinate with ThinkSpatial to transition shared files to a Uri-based file store
        //

        File storageDirectory = new File( "/storage" );
```

```
        File[] storageDirectoryFolders = storageDirectory.listFiles();


        File sdCardStorageDirectory = new File("" );


//        // Check all the files and folders within the storage parent directory
        for( File storageDirectoryFolder : storageDirectoryFolders ) {


            if ( storageDirectoryFolder.getAbsolutePath().matches( "(.*\\p{XDigit}{4}-\\p{XDigit}{4})" ) ){


                sdCardStorageDirectory = storageDirectoryFolder;


            }


        }


        File cullFile = new File( sdCardStorageDirectory,
"/Android/data/au.gov.gbrmpa.cots.capture/files/culldata.db");
        File surveillanceFile = new File( sdCardStorageDirectory,
"/Android/data/au.gov.gbrmpa.cots.surveillance/files/surveillance.db");
        File rhisFile = new File( sdCardStorageDirectory,
"/Android/data/au.gov.gbrmpa.cots.rhis/files/rhisdata.db");


        String DATABASE_PATH = "/data/data/" + "au.csiro.cotscontrolcentre_decisionsupporttool_0_0" + "/" +
"databases/";
        File dbFile = new File( DATABASE_PATH,"cotsData.sqlite");
        SQLiteDatabase db = SQLiteDatabase.openDatabase( dbFile.getAbsolutePath(), null,
SQLiteDatabase.OPEN_READWRITE );


        DiveList newCullData = loadCullData( cullFile, db );
        MantaList newMantaData = loadSurveillanceData( surveillanceFile, db );
//        List<Rhis> newRhisData = loadRhisData( rhisFile, db );


        for ( Dive dive: newCullData ) {


            if ( dive != null ){


                addCullDataToAppDatabase(dive, db);


            }


        }


        for ( Manta manta: newMantaData ) {


            if ( manta != null ) {


                addMantaDataToAppDatabase(manta, db);


            }


        }
```

```
        db.close();

    }

    private static DiveList loadCullData( File file, SQLiteDatabase db ){

        SQLiteDatabase divedb = SQLiteDatabase.openDatabase( file.getAbsolutePath(), null,
SQLiteDatabase.OPEN_READONLY );
        Cursor divedbCursor = divedb.rawQuery("SELECT * FROM culldata",null);
        DiveList returnDiveList = new DiveList();

        while ( divedbCursor.moveToNext() ) {

            // Load json
            String diveJSONString = divedbCursor.getString( 1 );

            try {

                JSONObject diveJSONObject = new JSONObject(diveJSONString);

                // Process json
                returnDiveList.add( convertJSONtoDive( diveJSONObject, db ) );

            } catch ( Throwable t ){

                Log.e("My App", "Could not parse malformed JSON: \"" + diveJSONString + "\"");

            }

        }

        divedb.close();

        return returnDiveList;

    }

    private static MantaList loadSurveillanceData( File file, SQLiteDatabase db ){

        SQLiteDatabase surveillancedb = SQLiteDatabase.openDatabase( file.getAbsolutePath(), null,
SQLiteDatabase.OPEN_READONLY );
        Cursor surveillancedbCursor = surveillancedb.rawQuery("SELECT * FROM surveillance",null);
        MantaList returnMantaList = new MantaList();

        while ( surveillancedbCursor.moveToNext() ) {

            // Load json
            String surveillanceJSONString = surveillancedbCursor.getString( 1 );

            try {
```

```
            JSONObject mantaJSONObject = new JSONObject(surveillanceJSONString);

            // Process json
            returnMantaList.add( convertJSONtoManta( mantaJSONObject, db ) );

        } catch ( Throwable t ){

            Log.e("My App", "Could not parse malformed JSON: \"" + surveillanceJSONString + "\"");

        }

    }

    surveillancedb.close();

    return returnMantaList;

}

// TODO: Need to load new Site definitions


///////////////    METHODS TO CONVERT THINKSPATIAL JSON INTO COTSCONTROLCENTRE CUSTOM TYPES

private static Dive convertJSONtoDive( JSONObject jsonObject, SQLiteDatabase db ){

    try {

        int diveId = 0; //Need to generate this as we add it to the SQLTable;
        String diveDate = jsonObject.getString("divedate").substring( 0, 10);;
        double diveAverageDepth = jsonObject.getDouble("depth");
        int diveBottomTime = jsonObject.getInt("bottomTime");
        int diveLessThanFifteenCentimetres = jsonObject.getJSONArray("cohorts").getInt(0);
        int diveFifteenToTwentyFiveCentimetres = jsonObject.getJSONArray("cohorts").getInt(1);
        int diveTwentyFiveToFortyCentimetres = jsonObject.getJSONArray("cohorts").getInt(2);
        int diveGreaterThanFortyCentimetres = jsonObject.getJSONArray("cohorts").getInt(3);
        // We don't really want to use the ThinkSpatial assignment of manta tows to Sites
        // because we don't know how they implemented it, it's assumptions or limitations. To
        // use our own method of assigning manta tows to their nearest Site we need the reef at
        // which the manta tow was collected, and the mean lat and long of the manta tow.
        //
        // We have an issue here because not all manta tows are assigned to cull sites by the
        // ThinkSpatial app, which is fair enough - but if they are not assigned to a cull site
        // the ThinkSpatial app also does not export which reef they are assigned to, with the
        // exception of the info of the internal ThinkSpatial reef id code, which is not fair
        // enough at all, and makes our job almost impossible. The best workarounds I can think
        // of right now are: 1) (fast) given that other mantas will generally be generated at
        // the same reef on the same import, I could maintain a list of the recent internal
        // ThinkSpatial reef_ids and actual reef names from other rows imported during the
```

```
            // current import, and use that to figure out the actual reef; or 2) (horrifically slow)
            // write a method that imports all reef polygons and checks which one the manta tow
            // intersects. Sigh.
            //
            //TODO: Compensate for ThinkSpatial's borked manta export when the manta is not assigned to a
Site
            //
            // The other thing we should do here is use our
            String siteName = jsonObject.getJSONObject("cullzone").getString("name");
            int siteId = findSiteIdFromSiteName( siteName, db );
            String vesselName = jsonObject.getJSONObject("voyage").getString("vessel");
            int vesselVoyage = jsonObject.getJSONObject("voyage").getInt("title");
            String voyageStartDate = jsonObject.getJSONObject("voyage").getString("start");
            String voyageStopDate = jsonObject.getJSONObject("voyage").getString("end");
            int voyageId = findVoyageIdFromVesselNameAndVoyageNumber( vesselName, vesselVoyage,
voyageStartDate, voyageStopDate, db );

            return new Dive( diveId, diveDate, diveAverageDepth, diveBottomTime,
diveLessThanFifteenCentimetres, diveFifteenToTwentyFiveCentimetres, diveTwentyFiveToFortyCentimetres,
diveGreaterThanFortyCentimetres, siteId, voyageId );

        } catch ( Throwable t ){

            Log.e("CCC_JSON", "Could not parse malformed JSON: \"" + jsonObject + "\"");

        }

        return null;

    }

    private static Manta convertJSONtoManta( JSONObject jsonObject, SQLiteDatabase db ){

        try {

            int mantaId = 0; //Need to generate this as we add it to the SQLTable;
            String mantaDate = jsonObject.getString("towDate").substring( 0, 10);
            JSONArray mantaTowLine = jsonObject.getJSONObject( "towLine" ).getJSONObject( "geometry"
).getJSONArray( "coordinates" );
            double mantaStartLat = ( (JSONArray) mantaTowLine.get(0) ).getDouble(1);
            double mantaStartLong =  ( (JSONArray) mantaTowLine.get(0) ).getDouble(0);
            double mantaStopLat =   ( (JSONArray) mantaTowLine.get(mantaTowLine.length() - 1)
).getDouble(1);
            double mantaStopLong = ( (JSONArray) mantaTowLine.get(mantaTowLine.length() - 1)
).getDouble(0);
            double mantaMeanLat = (mantaStartLat + mantaStopLat) / 2;
            double mantaMeanLong = (mantaStartLong + mantaStopLong) / 2;
            int mantaCots = jsonObject.getInt("cotsSeen");
            String mantaScars = jsonObject.getString("scarsSeen");
            String siteName = ( (JSONObject) jsonObject.getJSONArray("cullzones").get(0)
).getString("name");
            int siteId = findSiteIdFromSiteName( siteName, db );
            String vesselName = jsonObject.getJSONObject("voyage").getString("vessel");
            int vesselVoyage = jsonObject.getJSONObject("voyage").getInt("title");
```

```
            String voyageStartDate = jsonObject.getJSONObject("voyage").getString("start").substring( 0,
10);
            String voyageStopDate = jsonObject.getJSONObject("voyage").getString("end").substring( 0, 10);
            int voyageId = findVoyageIdFromVesselNameAndVoyageNumber( vesselName, vesselVoyage,
voyageStartDate, voyageStopDate, db );

            return new Manta( mantaId, mantaDate, mantaStartLat, mantaStartLong, mantaStopLat,
mantaStopLong, mantaMeanLat, mantaMeanLong, mantaCots, mantaScars, siteId, voyageId );

        } catch ( Throwable t ){

            Log.e("CCC_JSON", "Could not parse malformed JSON: \"" + jsonObject + "\"");

        }

        return null;

    }

    private static Rhis convertJSONtoRhis(JSONObject jsonObject, SQLiteDatabase db ){

        try {

            int rhisId = 0; //Need to generate this as we add it to the SQLTable;
            String rhisDate = jsonObject.getJSONObject("static").getString("surveydate").substring( 0, 10);
            double rhisCoralCover = jsonObject.getJSONObject("Benthos Observations").getInt("Live Coral");
            int rhisCOTSAdult = 0;
            int rhisCOTSJuvenile = 0;
            JSONArray predatorObservations = jsonObject.getJSONArray("Predator Observations");
            for ( int i = 0; i < predatorObservations.length(); i++ ){

                if( ( (JSONObject) predatorObservations.get(i) ).getJSONObject("Observations").equals(
"Adults" ) ) {

                    rhisCOTSAdult = ( (JSONObject) predatorObservations.get(i) ).getInt("COTS");

                };

                if( ( (JSONObject) predatorObservations.get(i) ).getJSONObject("Observations").equals(
"Juveniles" ) ) {

                    rhisCOTSJuvenile = ( (JSONObject) predatorObservations.get(i) ).getInt("COTS");

                };

            }
            String rhisVisibility = jsonObject.getJSONObject("static").getString("visibility");
            int siteId = 0; //Need to generate this as we add it to the SQLTable;

            return new Rhis( rhisId, rhisDate, rhisCoralCover, rhisCOTSAdult, rhisCOTSJuvenile,
rhisVisibility, siteId );
```

```
        } catch ( Throwable t ){

            Log.e("CCC_JSON", "Could not parse malformed JSON: \"" + jsonObject + "\"");

        }

        return null;

    }

    ///////////////    METHODS TO LOOK UP CURRENT DATABASE TO CROSS-REFERENCE

    private static int findSiteIdFromSiteName( String siteName, SQLiteDatabase db ) {

       String query =
                "SELECT " +
                        COTSDataContract.SiteEntry.SITE_TABLE_NAME + "." +
COTSDataContract.SiteEntry.SITE_TABLE_COLUMN_ID +
                        " FROM " +
                        COTSDataContract.SiteEntry.SITE_TABLE_NAME +
                        " WHERE " +
                        COTSDataContract.SiteEntry.SITE_TABLE_NAME + "." +
COTSDataContract.SiteEntry.SITE_TABLE_COLUMN_SITE_NAME + " = '" + siteName + "'";

        Cursor dbCursor = db.rawQuery( query,null);

        int returnSiteId = 0;

        // In theory, a cursor with either 0 rows, if the Site does not already exist, or at most
        // one row, if the Site does already exist, should be returned. If the Site does
        // not already exist, we create it and record the siteId of the new record. If it does,
        // we look up its siteId.
        while ( dbCursor.moveToNext() ) {

            returnSiteId = dbCursor.getInt( dbCursor.getColumnIndex(
COTSDataContract.SiteEntry.SITE_TABLE_COLUMN_ID  ) );

        }

        return returnSiteId;

    }

    private static int findVoyageIdFromVesselNameAndVoyageNumber( String vesselName, int voyageNumber,
String voyageStartDate, String voyageStopDate, SQLiteDatabase db ) {

        String query =
                "SELECT " +
                        COTSDataContract.VoyageEntry.VOYAGE_TABLE_NAME + "." +
COTSDataContract.VoyageEntry.VOYAGE_TABLE_COLUMN_ID +
                        " FROM " +
                        COTSDataContract.VoyageEntry.VOYAGE_TABLE_NAME +
```

```
                        " LEFT JOIN " +
                        COTSDataContract.VesselEntry.VESSEL_TABLE_NAME +
                        " ON " +
                        COTSDataContract.VesselEntry.VESSEL_TABLE_NAME + "." +
COTSDataContract.VesselEntry.VESSEL_TABLE_COLUMN_ID +
                        " = " +
                        COTSDataContract.VoyageEntry.VOYAGE_TABLE_NAME + "." +
COTSDataContract.VoyageEntry.VOYAGE_TABLE_COLUMN_ID +
                        " WHERE " +
                        COTSDataContract.VesselEntry.VESSEL_TABLE_NAME + "." +
COTSDataContract.VesselEntry.VESSEL_TABLE_COLUMN_VESSEL_NAME + " = '" + vesselName + "'" +
                        " AND " +
                        COTSDataContract.VoyageEntry.VOYAGE_TABLE_NAME + "." +
COTSDataContract.VoyageEntry.VOYAGE_TABLE_COLUMN_VOYAGE_NUMBER + " = " + voyageNumber;

        Cursor dbCursor = db.rawQuery( query,null);

        int returnVoyageId = 0;

        // In theory, a cursor with either 0 rows, if the voyage does not already exist, or at most
        // one row, if the voyage does already exist, should be returned. If the voyage does
        // not already exist, we create it and record the voyageId of the new record. If it does,
        // we look up its voyageId.
        if ( dbCursor.moveToNext() == false ) {

            String query2 =
                    "SELECT " +
                        COTSDataContract.VesselEntry.VESSEL_TABLE_NAME + "." +
COTSDataContract.VesselEntry.VESSEL_TABLE_COLUMN_ID +
                        " FROM " +
                        COTSDataContract.VesselEntry.VESSEL_TABLE_NAME +
                        " WHERE " +
                        COTSDataContract.VesselEntry.VESSEL_TABLE_NAME + "." +
COTSDataContract.VesselEntry.VESSEL_TABLE_COLUMN_VESSEL_NAME + " = '" + vesselName +"'";

            Cursor dbCursor2 = db.rawQuery( query2,null);

            int vesselId = 0;

            while ( dbCursor2.moveToNext() ) {

                vesselId = dbCursor2.getInt( dbCursor2.getColumnIndex(
COTSDataContract.VesselEntry.VESSEL_TABLE_COLUMN_ID  ) );

            }

            Voyage newVoyage = new Voyage( 0, voyageNumber, voyageStartDate, voyageStopDate, vesselId );

            returnVoyageId = (int) addVoyageDataToAppDatabase( newVoyage, db );

        } else {

            returnVoyageId = dbCursor.getInt( dbCursor.getColumnIndex(
COTSDataContract.VoyageEntry.VOYAGE_TABLE_COLUMN_ID  ) );
```

```
        }

        // In practice, however, we'd often expect the Voyage to not currently exist in the main
        // SQL database yet, so we have to add it, and get the new row number / id from the new
        // entry to link our Dive and Manta and RHIS records to the new voyage

        return returnVoyageId;

    }

    ///////////////    METHODS TO ADD NEW RECORDS TO SQLITE DATABASE
    //TODO: If manta can't be associated to a Site, don't enter it into the database

    private static long addVoyageDataToAppDatabase( Voyage voyage, SQLiteDatabase db ){

        // Create a new map of values, where column names are the keys
        ContentValues newVoyageEntryValues = new ContentValues();

        // mantaId: Note that the new __id field is created by adding the entry to the VoyageTable;
        newVoyageEntryValues.put( COTSDataContract.VoyageEntry.VOYAGE_TABLE_COLUMN_VOYAGE_NUMBER,
voyage.getVoyageNumber() );
        newVoyageEntryValues.put( COTSDataContract.VoyageEntry.VOYAGE_TABLE_COLUMN_START_DATE,
voyage.getStartDate() );
        newVoyageEntryValues.put( COTSDataContract.VoyageEntry.VOYAGE_TABLE_COLUMN_STOP_DATE,
voyage.getStopDate() );
        newVoyageEntryValues.put( COTSDataContract.VoyageEntry.VOYAGE_TABLE_COLUMN_VESSEL_ID,
voyage.getVesselId() );

// Insert the new row, returning the primary key value of the new row
        long newRowId = db.insert( COTSDataContract.VoyageEntry.VOYAGE_TABLE_NAME, null,
newVoyageEntryValues);

        return newRowId;

    }

    private static long addCullDataToAppDatabase( Dive dive, SQLiteDatabase db ){

        // Load db

        // Create a new map of values, where column names are the keys
        ContentValues newDiveEntryValues = new ContentValues();

        // diveId: Note that the new __id field is created by adding the entry to the DiveTable;
        newDiveEntryValues.put( COTSDataContract.DiveEntry.DIVE_TABLE_COLUMN_DATE, dive.getDiveDate() );
        newDiveEntryValues.put( COTSDataContract.DiveEntry.DIVE_TABLE_COLUMN_AVERAGE_DEPTH,
dive.getDiveDate() );
        newDiveEntryValues.put( COTSDataContract.DiveEntry.DIVE_TABLE_COLUMN_BOTTOM_TIME,
dive.getDiveDate() );
        newDiveEntryValues.put( COTSDataContract.DiveEntry.DIVE_TABLE_COLUMN_LESS_THAN_FIFTEEN_CENTIMETRES,
dive.getDiveDate() );
        newDiveEntryValues.put(
COTSDataContract.DiveEntry.DIVE_TABLE_COLUMN_FIFTEEN_TO_TWENTY_FIVE_CENTIMETRES, dive.getDiveDate() );
```

```
        newDiveEntryValues.put(
COTSDataContract.DiveEntry.DIVE_TABLE_COLUMN_TWENTY_FIVE_TO_FORTY_CENTIMETRES, dive.getDiveDate() );

        newDiveEntryValues.put(
COTSDataContract.DiveEntry.DIVE_TABLE_COLUMN_GREATER_THAN_FORTY_CENTIMETRES, dive.getDiveDate() );

        newDiveEntryValues.put( COTSDataContract.DiveEntry.DIVE_TABLE_COLUMN_SITE_ID, dive.getSiteId() );

        newDiveEntryValues.put( COTSDataContract.DiveEntry.DIVE_TABLE_COLUMN_VOYAGE_ID, dive.getVoyageId()
);


// Insert the new row, returning the primary key value of the new row

        long newRowId = db.insert( COTSDataContract.DiveEntry.DIVE_TABLE_NAME, null, newDiveEntryValues);


        return newRowId;


    }


    private static long addMantaDataToAppDatabase( Manta manta, SQLiteDatabase db ){


        // Create a new map of values, where column names are the keys

        ContentValues newMantaEntryValues = new ContentValues();


        // mantaId: Note that the new __id field is created by adding the entry to the MantaTable;

        newMantaEntryValues.put( COTSDataContract.MantaEntry.MANTA_TABLE_COLUMN_DATE, manta.getMantaDate()
);

        newMantaEntryValues.put( COTSDataContract.MantaEntry.MANTA_TABLE_COLUMN_START_LAT,
manta.getMantaStartLat() );

        newMantaEntryValues.put( COTSDataContract.MantaEntry.MANTA_TABLE_COLUMN_START_LONG,
manta.getMantaStartLong() );

        newMantaEntryValues.put( COTSDataContract.MantaEntry.MANTA_TABLE_COLUMN_STOP_LAT,
manta.getMantaStopLat() );

        newMantaEntryValues.put( COTSDataContract.MantaEntry.MANTA_TABLE_COLUMN_STOP_LONG,
manta.getMantaStopLong() );

        newMantaEntryValues.put( COTSDataContract.MantaEntry.MANTA_TABLE_COLUMN_MEAN_LAT,
manta.getMantaMeanLat() );

        newMantaEntryValues.put( COTSDataContract.MantaEntry.MANTA_TABLE_COLUMN_MEAN_LONG,
manta.getMantaMeanLong() );

        newMantaEntryValues.put( COTSDataContract.MantaEntry.MANTA_TABLE_COLUMN_COTS, manta.getMantaCOTS()
);

        newMantaEntryValues.put( COTSDataContract.MantaEntry.MANTA_TABLE_COLUMN_SCARS,
manta.getMantaScars() );

        newMantaEntryValues.put( COTSDataContract.MantaEntry.MANTA_TABLE_COLUMN_SITE_ID, manta.getSiteId()
);

        newMantaEntryValues.put( COTSDataContract.MantaEntry.MANTA_TABLE_COLUMN_VOYAGE_ID,
manta.getVoyageId() );


        //TODO: Test whether entry is already in database


// Insert the new row, returning the primary key value of the new row

        long newRowId = db.insert( COTSDataContract.MantaEntry.MANTA_TABLE_NAME, null,
newMantaEntryValues);


        return newRowId;



    }


    private static long addRhisDataToAppDatabase( Rhis rhis, SQLiteDatabase db ){
```

```
        // Create a new map of values, where column names are the keys
        ContentValues newRhisEntryValues = new ContentValues();

        // rhisId: Note that the new __id field is created by adding the entry to the DiveTable;
        newRhisEntryValues.put( COTSDataContract.RhisEntry.RHIS_TABLE_COLUMN_DATE, rhis.getRhisDate() );
        newRhisEntryValues.put( COTSDataContract.RhisEntry.RHIS_TABLE_COLUMN_AVERAGE_CORAL_COVER,
rhis.getRhisCoralCover() );
        newRhisEntryValues.put( COTSDataContract.RhisEntry.RHIS_TABLE_COLUMN_COTS_ADULTS,
rhis.getRhisCOTSAdults() );
        newRhisEntryValues.put( COTSDataContract.RhisEntry.RHIS_TABLE_COLUMN_COTS_JUVENILES,
rhis.getRhisCOTSJuveniles() );
        newRhisEntryValues.put( COTSDataContract.RhisEntry.RHIS_TABLE_COLUMN_VISIBILITY,
rhis.getRhisVisibility() );
        // siteId: Note that the new linked siteId field is created by cross referencing with the
SiteTable;

// Insert the new row, returning the primary key value of the new row
        long newRowId = db.insert( COTSDataContract.RhisEntry.RHIS_TABLE_NAME, null, newRhisEntryValues);

        return newRowId;

    }

}
```

## B.5 ImplementDecisionTreeAtReef.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.model;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collections;
import java.util.Date;
import java.util.List;
import java.util.concurrent.TimeUnit;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.MainActivity;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Dive;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.DiveList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Manta;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.MantaList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Site;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.SiteList;

//
// This class contains the logic required to implement the decision tree at a Reef. When a new
// instance is created, it must be passed:
//      1) the name of the Reef;
//      2) the siteList of all Sites at the Reef;
//      3) the diveList of all Dives at the Reef since the last Manta tow
//      4) the most recent Mantas at the Reef
```

```
//
// The goal of the class is to ascertain:
// 1) Whether a Reef is in Maintenance or Intensive Control Mode
// 2) Whether the Reef needs to be manta towed
// 3) Whether the Reef needs to be culled
// 4) If so, in what order should the Sites be culled
//
// These decisions are underpinned by the logic contained in the "simple decision tree" in the
// reports Fletcher et al. (2020) and Fletcher (2020).
//
//

public class ImplementDecisionTreeAtReef {

    private String _reefName;
    private SiteList _siteList;
    private DiveList _diveList;
    private MantaList _mantaList;

    // Empty constructor
    public ImplementDecisionTreeAtReef(){
    }

    public ImplementDecisionTreeAtReef( String reefName, SiteList siteList, DiveList diveList, MantaList
mantaList ){

        this._reefName = reefName;
        this._siteList = siteList;
        this._diveList = diveList;
        this._mantaList = mantaList;

    }

    public String ImplementDecisionTreeAtReefAndFindControlTasks(){

        // Check whether Reef is in maintenance mode or intensive mode

        String returnString = "";

        if ( maintenanceModeQ() ) {

            // If Maintenance Reef
            returnString = provide_MaintenanceReef_Tasks();

        } else {

            // If Intensive Control Reef
            returnString = provide_IntensiveControlReef_Tasks();

        }
```

```
        return returnString;

    }

    public List<Integer> ImplementDecisionTreeAtReefAndFindSiteIdsToBeControlledInOrder(){

        List<Integer> returnSiteIdList = new ArrayList<>();

        if ( maintenanceModeQ() ) {

            // If Maintenance Reef don't return anything

        } else {

            // If Intensive Control Reef, return the SiteIds of Sites to be controlled in order
            returnSiteIdList = provide_IntensiveControlReef_SiteIds( true );

        }

        return returnSiteIdList;

    }

    private String provide_IntensiveControlReef_Tasks() {

        String returnIntensiveControlReefTasks = "";

        List<Integer> intensiveControlReefSiteIds = provide_IntensiveControlReef_SiteIds( true );

        if ( intensiveControlReefSiteIds.isEmpty() ) {

            returnIntensiveControlReefTasks = "All available Sites culled, move to next Reef.";

        } else if ( intensiveControlReefSiteIds.get(0) == -1 ) {

            returnIntensiveControlReefTasks = "Manta tow due. Manta tow Reef before generating Workplan.";

        } else {

            int i = 1;

            returnIntensiveControlReefTasks = "Begin culling sites in the following order: \n\n";

            for ( Integer siteId : intensiveControlReefSiteIds ) {

                Site site = this._siteList.getSiteBySiteId( siteId );

                returnIntensiveControlReefTasks = returnIntensiveControlReefTasks + Integer.toString(i) +
": Site " + site.getSiteName() + " \n\n";

                i++;
```

```
            }

        }

        return returnIntensiveControlReefTasks;

    }


    private List<Integer> provide_IntensiveControlReef_SiteIds( boolean forceDespiteManta ) {

        List<Integer> returnIntensiveControlReefSiteIds = new ArrayList<>();

        boolean mantaTowRequired = true;

        // We provide the user the option to force the calculation of Site orders even if a Manta
        // is required.
        if ( forceDespiteManta ) {

            mantaTowRequired = false;

        } else if ( this._mantaList.isEmpty() ) { // Check if this is the first ever Voyage to Reef by
checking whether there are any Mantas

            // If so, flag manta tow required
            mantaTowRequired = true;

        } else {

            // Check whether Reef has been manta towed within the manta tow period.
            mantaTowRequired = ( numberOfDaysSinceLastManta() >
MainActivity.daysToMantaTowIntensiveControlReef );

        }

        // If manta tow is required, we'll flag that as the first action, after which additional
        // input will be provided, but we don't provide any other advice right now.
        if ( mantaTowRequired ) {

            returnIntensiveControlReefSiteIds.add( -1 );

            // Don't do anything and return a null List

        } else { // If manta tow is not required

            List<Integer> siteIdsOfSitesToBeCulled = new ArrayList<>();
            List<Dive> diveSitesToBeCulledBasedOnLastCull = new ArrayList<>();
            List<Manta> mantaSitesToBeCulledBasedOnLastManta = new ArrayList<>();

            // Iterate through each Site at the Reef
```

```
            for ( Site site : this._siteList ) {

                // Get the mantas: 1) from the most recent Voyage during which Mantas were
                // collected; 2) at the current Site. If no Mantas have been conducted at this Reef,
                // or non assigned to this Site, the result will be an empty list.
                // NOTE THAT THE ORDER OF THESE COMMANDS IS LOGICALLY IMPORTANT
                MantaList mostRecentMantasAtSite =
this._mantaList.getMantasOnMostRecentVoyage().getMantasBySiteId(site.getSiteId());

                // Get the most recent Dives at this Site from the most recent Voyage during which
                // a Dive was collected at at this Site.
                // NOTE THAT THE ORDER OF THESE COMMANDS IS LOGICALLY IMPORTANT
                DiveList mostRecentDivesAtSite =
this._diveList.getDivesBySiteId(site.getSiteId()).getDivesOnMostRecentVoyage();

                // Now, if: 1) there have been Dives at the Site; then 2) find the Date of most recent
                // Dive at the Site, and 3) if there was also a manta at the Site; 3a) find out if
                // the Dive is more recent than the most recent Manta tow, then 3b) if the Dive
                // was conducted more than the site revisitation schedule ago, then 3c) add the
                // Dives to diveSitesToBeCulledBasedOnLastCull; otherwise, if 4) there was never
                // any Manta conducted at the Site (which should never be the case), 4a) if the Dive
                // was conducted more than the site revisitation schedule ago, then 4b) add the
                // Dives to diveSitesToBeCulledBasedOnLastCull
                if (!mostRecentDivesAtSite.isEmpty()) {

                    // Create a composite Dive from however many Dives occurred at the Site during
                    // the most recent Voyage
                    Dive compositeDive = mostRecentDivesAtSite.createCompositeDive();

                    if (!mostRecentMantasAtSite.isEmpty()) { // If there are both Mantas and Dives at the
Site

                        // Create composite Manta
                        Manta compositeManta = mostRecentMantasAtSite.createCompositeManta();

                        if (compositeDive.getDiveDateAsDate().after(compositeManta.getMantaDateAsDate())) {
// If the Dive is more recent than the Manta

                            if (compositeDive.numberOfDaysSinceDive() >
MainActivity.daysToCullSiteAtIntensiveControlReef) { // If the Dive was Dived long enough ago

                                if ( !compositeDive.belowEcologicalThreshold() ) { // Check if the Site is
above the Ecological Threshold

                                    diveSitesToBeCulledBasedOnLastCull.add(compositeDive);

                                }

                            } // Else the Site has been Dived too recently, so do nothing

                        } else { // If the Manta is more recent than the Dive
```

```
                            if ( !compositeManta.belowEcologicalThreshold() ) { // Check if the Site is
above the Ecological Threshold

                                mantaSitesToBeCulledBasedOnLastManta.add(compositeManta); // If so, add it
to the list

                            }
                        }

                    } else { // If there are only Dives at the Site (which should never happen, but who
knows)



                        if ( compositeDive.numberOfDaysSinceDive() >
MainActivity.daysToCullSiteAtIntensiveControlReef ) { // If the Dive was Dived long enough ago

                            if ( !compositeDive.belowEcologicalThreshold() ) { // Check if the Site is
above the Ecological Threshold

                                diveSitesToBeCulledBasedOnLastCull.add(compositeDive);

                            }

                        } // Else the Site has been Dived too recently, so do nothing

                    }

                } else { // Else if there are no Dives at the Site

                    if ( !mostRecentMantasAtSite.isEmpty() ) { // If there is a Manta at the Site

                        // Create composite Manta
                        Manta compositeManta = mostRecentMantasAtSite.createCompositeManta();

                        if ( !compositeManta.belowEcologicalThreshold() ) { // Check if the Site is above
the Ecological Threshold


mantaSitesToBeCulledBasedOnLastManta.add(mostRecentMantasAtSite.createCompositeManta());

                        }
                    }

                }

            }

            // Now, sort the two lists, highest COTS cull numbers and highest Mantas first
            Collections.sort( diveSitesToBeCulledBasedOnLastCull, Dive.getDiveCullNumberComparator() );
            Collections.sort( mantaSitesToBeCulledBasedOnLastManta, Manta.getMantaCOTSNumberComparator() );

            // Then add the siteIds of the Dives first, then the Mantas
            for ( Dive dive : diveSitesToBeCulledBasedOnLastCull ){
```

```
                siteIdsOfSitesToBeCulled.add ( dive.getSiteId() );

            }

            for ( Manta manta : mantaSitesToBeCulledBasedOnLastManta ){

                siteIdsOfSitesToBeCulled.add ( manta.getSiteId() );

            }

            if ( siteIdsOfSitesToBeCulled.isEmpty() ) {

                // Don't provide anything

            } else {

                returnIntensiveControlReefSiteIds = siteIdsOfSitesToBeCulled;

            }

        }

        return returnIntensiveControlReefSiteIds;

    }


    private String provide_MaintenanceReef_Tasks() {

        String returnString = "";

        boolean mantaTowRequired = true;

        List<Manta> mantaListCopy = this._mantaList.getMantaListCopy();

        Collections.sort( mantaListCopy, Manta.getMantaDateComparator() );

        Date latestMantaDate = mantaListCopy.get(0).getMantaDateAsDate();

        Calendar todaysDate = Calendar.getInstance();
        long timeSinceLastManta = todaysDate.getTime().getTime() - latestMantaDate.getTime();
        long numberOfDaysSinceLastManta = TimeUnit.MILLISECONDS.toDays( timeSinceLastManta );

        mantaTowRequired = ( numberOfDaysSinceLastManta > MainActivity.daysToMantaTowMaintenanceReef );

        if ( mantaTowRequired ) {

           returnString = "Manta tow out of date, perform comprehensive manta tow.";
```

```
        } else {

            returnString = "No manta tow required, move to next Reef.";

        }

        return returnString;

    }



    //
    // This function can be coded more efficiently by returning as soon as the conditions for
    // Maintenance Mode are not achieved, however we leave it like this for now because it's
    // readable. We can work on efficiency later
    //
    private boolean maintenanceModeQ (){


//        HashMap<Integer, Integer> mostRecentDiveIdAtEachSiteId =
this._siteList.getMostRecentDiveAtEachSite( this._diveList );
//        HashMap<Integer, List<Integer>> mostRecentMantaIdsAtEachSiteId =
this._siteList.getMostRecentMantasAtEachSite( this._mantaList );


        boolean returnMaintenanceModeQ = true;

        // Check if the Reef has no Mantas or Dives
        if ( this._diveList.isEmpty() && this._mantaList.isEmpty() ) {

           // If no, Reef is in Intensive Management Mode
           returnMaintenanceModeQ = true;

        } else { // Reef has at least one manta or cull

           boolean allSitesBelowEcologicalThreshold = true;

           // Check to see whether each Site at the Reef is below the Ecological Threshold by either
           // cull or Manta, depending on which is more recent
            for ( Site site : this._siteList ) {

                // Get the mantas: 1) from the most recent Voyage during which Mantas were
                // collected; 2) at the current Site. If no Mantas have been conducted at this Reef,
                // or non assigned to this Site, the result will be an empty list.
                // NOTE THAT THE ORDER OF THESE COMMANDS IS LOGICALLY IMPORTANT
                MantaList mostRecentMantasAtSite =
this._mantaList.getMantasOnMostRecentVoyage().getMantasBySiteId(site.getSiteId());

                // Get the most recent Dives at this Site from the most recent Voyage during which
                // a Dive was collected at at this Site.
                // NOTE THAT THE ORDER OF THESE COMMANDS IS LOGICALLY IMPORTANT
                DiveList mostRecentDivesAtSite =
this._diveList.getDivesBySiteId(site.getSiteId()).getDivesOnMostRecentVoyage();
```

```
                // Now, if: 1) there have been Dives at the Site; then 2) find the Date of most recent
                // Dive at the Site, and 3) if there was also a manta at the Site; 3a) find out if
                // the Dive is more recent than the most recent Manta tow, then 3b) if the Dive
                // was conducted more than the site revisitation schedule ago, then 3c) add the
                // Dives to diveSitesToBeCulledBasedOnLastCull; otherwise, if 4) there was never
                // any Manta conducted at the Site (which should never be the case), 4a) if the Dive
                // was conducted more than the site revisitation schedule ago, then 4b) add the
                // Dives to diveSitesToBeCulledBasedOnLastCull
                if (!mostRecentDivesAtSite.isEmpty()) {

                    // Create a composite Dive from however many Dives occurred at the Site during
                    // the most recent Voyage
                    Dive compositeDive = mostRecentDivesAtSite.createCompositeDive();

                    if (!mostRecentMantasAtSite.isEmpty()) { // If there are both Mantas and Dives at the
Site

                        // Create composite Manta
                        Manta compositeManta = mostRecentMantasAtSite.createCompositeManta();

                        if (compositeDive.getDiveDateAsDate().after(compositeManta.getMantaDateAsDate())) {
// If the Dive is more recent than the Manta

                            // Check if the CPUE exceeded the Ecological Threshold
                            allSitesBelowEcologicalThreshold = allSitesBelowEcologicalThreshold &&
compositeDive.belowEcologicalThreshold();

                        } else { // If the Manta is more recent than the Dive, then add the Site as part of
the Manta list

                            // Check if the Manta exceeded the Ecological Threshold
                            allSitesBelowEcologicalThreshold = allSitesBelowEcologicalThreshold &&
compositeManta.belowEcologicalThreshold();

                        }

                    } else { // If there are only Dives at the Site (which should never happen, but who
knows)

                        // Check if the CPUE exceeded the Ecological Threshold
                        allSitesBelowEcologicalThreshold = allSitesBelowEcologicalThreshold &&
compositeDive.belowEcologicalThreshold();

                    }

                } else { // Else if there are no Dives at the Site

                    // Create composite Manta
                    Manta compositeManta = mostRecentMantasAtSite.createCompositeManta();

                    // Check if the Manta exceeded the Ecological Threshold
                    allSitesBelowEcologicalThreshold = allSitesBelowEcologicalThreshold &&
compositeManta.belowEcologicalThreshold();
```

```
                }

            }

            // Check if all Sites are below the Ecological Threshold

           if ( allSitesBelowEcologicalThreshold ) {

               // If every Site is below the Ecological Threshold, then the Reef qualifies for Maintenance
Mode
               returnMaintenanceModeQ = true;

           } else {

               // If not every Site is below the Ecological Threshold, then the Reef qualifies for
Intensive Control Mode
               returnMaintenanceModeQ = false;

           }

        }

        // Return result
        return returnMaintenanceModeQ;

    }

    private long numberOfDaysSinceLastManta(){

        // In theory,
        // we should have been passed only the most recent Manta tows at this Reef, but
        // because this is not enforced, we test the mantaList and extract only the most
        // recent mantas from the most recent Voyage at which a Manta was completed.
        Date latestMantaDate = this._mantaList.getMostRecentMantaDate();

        Calendar todaysDate = Calendar.getInstance();
        long timeSinceLastManta = todaysDate.getTime().getTime() - latestMantaDate.getTime();
        long numberOfDaysSinceLastManta = TimeUnit.MILLISECONDS.toDays( timeSinceLastManta );

        return numberOfDaysSinceLastManta;

    }

}
```

## B.6 FindMantaNearestSite.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0;

import android.app.Activity;
import android.content.Context;
```

```
import android.util.Log;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.Random;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Manta;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.MantaList;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Reef;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.SiteList;

import static java.lang.Math.floor;
import static java.lang.Math.max;
import static java.lang.Math.min;

public class FindMantaNearestSite {

    public static String findRandomMantaSite( Context context, Reef reef, SiteList siteList, MantaList
mantaList ) {

        int mantaRow = new Random().nextInt(mantaList.size());

        Manta manta = mantaList.getMantaByIndex(mantaRow);

        int reefId = reef.getReefId();

        double mantaMeanLatitude = manta.getMantaMeanLat();
        double mantaMeanLongitude = manta.getMantaMeanLong();

        int siteId = findNearestSiteId( context, reefId, mantaMeanLatitude, mantaMeanLongitude );

        String returnTest = "Recorded nearest siteId = " + Integer.toString( manta.getSiteId() ) + ",
calculated nearest siteId = " + Integer.toString( siteId );

        return returnTest;

    }

    public static int findNearestSiteId( Context context, int reefId, double latitude, double longitude ) {

        // Read data from file
        int resourceId = ( (Activity) context).getResources().getIdentifier( "r" + Integer.toString( reefId
), "raw", "au.csiro.cotscontrolcentre_decisionsupporttool_0_0" );
        InputStream is = ( (Activity) context).getResources().openRawResource( resourceId );
        BufferedReader reader = new BufferedReader(
                new InputStreamReader(is, Charset.forName("UTF-8")));
        String line = "";
```

```
ArrayList<ArrayList<Integer>> returnArray = new ArrayList<>();

// Read the headers
String reefID;
String reefName;
double startLat = 0;
double startLong = 0;
double resolution = 1;
int rows = 0;
int columns = 0;

try {

    // Read the headers
    reefID = findHeaderValue( reader.readLine() );
    reefName = findHeaderValue( reader.readLine() );
    startLat = Double.parseDouble( findHeaderValue( reader.readLine() ) ) ;
    startLong = Double.parseDouble( findHeaderValue( reader.readLine() ) );
    resolution = Double.parseDouble( findHeaderValue( reader.readLine() ) );
    rows = Integer.parseInt( findHeaderValue( reader.readLine() ) );
    columns = Integer.parseInt( findHeaderValue( reader.readLine() ) );

    int i = 0;

    while ( ( line = reader.readLine() ) != null ) {

        returnArray.add( new ArrayList<Integer>() );

        // Split the line into values using the comma as a separator
        String[] strArray = line.split(",");

        for (String str : strArray) {

            returnArray.get( i ).add( Integer.parseInt(str) );

        }

        i++;

    }

} catch ( IOException e1 ) {

    Log.e("ReadCSVFile", "Error" + line, e1);

    e1.printStackTrace();

}

// Load data into array structure
```

```
//        ArrayList<ArrayList<Integer>> nearestSiteArray = readNearestSiteArrayFromCSV( context, reefId );

        // Compared to the original Mathematica table, this process has:
        // 1) Loaded rows into rows and columns into columns;
        // 2) Loaded the data into the array top-to-bottom, in terms of the sequence of rows;
        // 3) Loaded the data in the same left-to-right order, in terms a single row
        //
        // That is, everything is loaded as it would display in the Mathematica file.
        //
        // The original Mathematica table had the southernmost latitude in row 1, and the
        // westernmost longitude in the first column. startLat refers to this southernmost latitude.
        // startLong refers to the westernmost longitude.
        //
        // Therefore, when we calculate our latitude index, we need to count forwards from the first
        // row. When we calculate our longitude index, we can count forward from the 0th column.

        // Calculate row
        int rowIndex = (int) min( floor( ( latitude - startLat ) / resolution ), rows );

        // Calculate column
        int columnIndex = (int) min( floor( ( longitude - startLong ) / resolution ), columns );

        int returnRowColumn = returnArray.get( rowIndex ).get( columnIndex );

        // Return answer
        return returnRowColumn;

    }

    private static String findHeaderValue( String line ){

        String[] lineSplit = line.split(",");

        String returnString = "";

        int i = 0;

        for ( String str : lineSplit ){

            if ( i > 0 ){ returnString = str; }

            i++;

        }

        return returnString;

    }

    private static ArrayList<ArrayList<Integer>> readNearestSiteArrayFromCSV(Context context, int reefId )
{
```

```
        int resourceId = ( (Activity) context).getResources().getIdentifier( "r" + Integer.toString( reefId
), "raw", "au.csiro.cotscontrolcentre_decisionsupporttool_0_0" );
        InputStream is = ( (Activity) context).getResources().openRawResource( resourceId );
        BufferedReader reader = new BufferedReader(
                new InputStreamReader(is, Charset.forName("UTF-8")));
        String line = "";
        ArrayList<Integer> arrayLine = new ArrayList<>();
        ArrayList<ArrayList<Integer>> returnArray = new ArrayList<>();

        try {

            // Read past the headers
            for ( int i = 1; i <= 7; i++ ){
                reader.readLine();
            }

            while ( ( line = reader.readLine() ) != null ) {

                arrayLine.clear();

                // Split the line into values using the comma as a separator
                String[] strArray = line.split(",");

                for (String str : strArray) {

                    arrayLine.add( Integer.parseInt(str) );

                }

                returnArray.add(arrayLine);

            }

        } catch ( IOException e1 ) {

            Log.e("ReadCSVFile", "Error" + line, e1);

            e1.printStackTrace();

        }

        return returnArray;

    }

}
```

## B.7 COTSDataContract.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data;

import android.net.Uri;
import android.provider.BaseColumns;

/**
 * Created by fle125 on 27/03/2017.
 */

public class COTSDataContract {

    public static final String CONTENT_AUTHORITY = "au.csiro.cotscontrolcentre_decisionsupporttool_0_0";

    public static final Uri BASE_CONTENT_URI = Uri.parse("content://" + CONTENT_AUTHORITY);

    public static final String PATH_REEFS = "reefs";
    public static final String PATH_REEF_POLYGONS = "reefpolygons";
    public static final String PATH_SITES = "sites";
    public static final String PATH_SITE_POLYGONS = "sitepolygons";
    public static final String PATH_VESSELS = "vessels";
    public static final String PATH_VOYAGES = "voyages";
    public static final String PATH_DIVES = "dives";
    public static final String PATH_MANTAS = "mantas";
    public static final String PATH_RHIS = "rhis";

    public static final String PATH_CONTROLLED = "controlled";

    public static final String PATH_WITH_CHARACTERISTIC = "with";
    public static final String PATH_BY_MEMBER_OF = "in";
    public static final String PATH_BY_CONTAINS = "contains";

    public static final String PATH_MODIFIER_ID = "id";
    public static final String PATH_MODIFIER_SITENAME = "sitename";

    public static final String PATH_MODIFIER_MOST_RECENT = "mostrecent";
    public static final String PATH_MODIFIER_AT_DATE = "atdate";

    public static final String PATH_NUMERIC_QUERY = "#";
    public static final String PATH_STRING_QUERY = "*";

    //        Inner class that defines the table contents of the Reef Table
    public static final class ReefEntry implements BaseColumns {

        //      The base CONTENT_URI used to query the Reefs from the Reef table from the content provider
        public static final Uri CONTENT_URI = BASE_CONTENT_URI.buildUpon()
                .appendPath(PATH_REEFS)
                .build();
```

```
        public static final String REEF_TABLE_NAME = "reefTable";

        public static final String REEF_TABLE_COLUMN_ID = "__reefId";
        public static final String REEF_TABLE_COLUMN_REEF_NAME = "reefReefName";
        public static final String REEF_TABLE_COLUMN_REEF_ID = "reefReefId";
        public static final String REEF_TABLE_COLUMN_LATITUDE = "reefLatitude";
        public static final String REEF_TABLE_COLUMN_LONGITUDE = "reefLongitude";

    }

    public static final class ReefPolygonsEntry implements BaseColumns {

        //      The base CONTENT_URI used to query the Reefs from the Reef table from the content provider
        public static final Uri CONTENT_URI = BASE_CONTENT_URI.buildUpon()
                .appendPath(PATH_REEF_POLYGONS)
                .build();

        public static final String REEF_POLYGONS_TABLE_NAME = "reefPolygonsTable";

        public static final String REEF_POLYGONS_TABLE_COLUMN_ID = "__reefPolygonsId";
        public static final String REEF_POLYGONS_TABLE_COLUMN_POINT_ORDER = "reefPolygonPointOrder";
        public static final String REEF_POLYGONS_TABLE_COLUMN_POINT_LATITUDE = "reefPolygonPointLatitude";
        public static final String REEF_POLYGONS_TABLE_COLUMN_POINT_LONGITUDE =
"reefPolygonPointLongitude";
        public static final String REEF_POLYGONS_TABLE_COLUMN_REEF_ID = "__reefId";

    }

    //    Inner class that defines the table contents of the Site table
    public static final class SiteEntry implements BaseColumns {

        //      The base CONTENT_URI used to query the Sites from the Dive table from the content provider
        public static final Uri CONTENT_URI = BASE_CONTENT_URI.buildUpon()
                .appendPath(PATH_SITES)
                .build();

        public static final String SITE_TABLE_NAME = "siteTable";

        public static final String SITE_TABLE_COLUMN_ID = "__siteId";
        public static final String SITE_TABLE_COLUMN_SITE_NAME = "siteSiteName";
        public static final String SITE_TABLE_COLUMN_LATITUDE = "siteLatitude";
        public static final String SITE_TABLE_COLUMN_LONGITUDE = "siteLongitude";
        public static final String SITE_TABLE_COLUMN_REEF_ID = "__reefId";
        public static final String SITE_TABLE_COLUMN_DIVE_IDS_OF_DIVES_AT_SITE = "diveIdsOfDivesAtSite";
        public static final String SITE_TABLE_COLUMN_MANTA_IDS_OF_MANTAS_AT_SITE =
"mantaIdsOfMantasAtSite";

    }

    public static final class SitePolygonsEntry implements BaseColumns {
```

```
        //      The base CONTENT_URI used to query the Reefs from the Reef table from the content provider
        public static final Uri CONTENT_URI = BASE_CONTENT_URI.buildUpon()
                .appendPath(PATH_SITE_POLYGONS)
                .build();

        public static final String SITE_POLYGONS_TABLE_NAME = "sitePolygonsTable";

        public static final String SITE_POLYGONS_TABLE_COLUMN_ID = "__sitePolygonsId";
        public static final String SITE_POLYGONS_TABLE_COLUMN_POINT_ORDER = "sitePolygonPointOrder";
        public static final String SITE_POLYGONS_TABLE_COLUMN_POINT_LATITUDE = "sitePolygonPointLatitude";
        public static final String SITE_POLYGONS_TABLE_COLUMN_POINT_LONGITUDE =
"sitePolygonPointLongitude";
        public static final String SITE_POLYGONS_TABLE_COLUMN_SITE_ID = "__siteId";

    }

    //    Inner class that defines the table contents of the Site table
    public static final class VesselEntry implements BaseColumns {

        //      The base CONTENT_URI used to query the Sites from the Dive table from the content provider
        public static final Uri CONTENT_URI = BASE_CONTENT_URI.buildUpon()
                .appendPath(PATH_VESSELS)
                .build();

        public static final String VESSEL_TABLE_NAME = "vesselTable";

        public static final String VESSEL_TABLE_COLUMN_ID = "__vesselId";
        public static final String VESSEL_TABLE_COLUMN_VESSEL_NAME = "vesselName";
        public static final String VESSEL_TABLE_COLUMN_VESSEL_SHORT_NAME = "vesselShortName";

    }

    /* Inner class that defines the table contents of the voyage table */
    public static final class VoyageEntry implements BaseColumns {

        /* The base CONTENT_URI used to query the Voyage table from the content provider */
        public static final Uri CONTENT_URI = BASE_CONTENT_URI.buildUpon()
                .appendPath(PATH_VOYAGES)
                .build();

        public static final String VOYAGE_TABLE_NAME = "voyageTable";

        public static final String VOYAGE_TABLE_COLUMN_ID = "__voyageId";
        public static final String VOYAGE_TABLE_COLUMN_VOYAGE_NUMBER = "voyageVesselVoyageNumber";
        public static final String VOYAGE_TABLE_COLUMN_START_DATE = "voyageStartDate";
        public static final String VOYAGE_TABLE_COLUMN_STOP_DATE = "voyageStopDate";
        public static final String VOYAGE_TABLE_COLUMN_VESSEL_ID = "__vesselId";
        // The column names below represent SQL calculation columns, not columns in the data table
        public static final String VOYAGE_TABLE_COLUMN_REEF_IDS_OF_REEFS_DURING_VOYAGE =
"reefIdsOfReefsDuringVoyage";
```

```
        public static final String VOYAGE_TABLE_COLUMN_SITE_IDS_OF_SITES_DURING_VOYAGE =
"siteIdsOfReefsDuringVoyage";

        public static final String VOYAGE_TABLE_COLUMN_DIVE_IDS_OF_DIVES_DURING_VOYAGE =
"diveIdsOfDivesDuringVoyage";

        public static final String VOYAGE_TABLE_COLUMN_MANTA_IDS_OF_MANTAS_DURING_VOYAGE =
"mantaIdsOfMantasDuringVoyage";


    }



    /* Inner class that defines the table contents of the dive table */
    public static final class DiveEntry implements BaseColumns {


        /* The base CONTENT_URI used to query the Dive table from the content provider */
        public static final Uri CONTENT_URI = BASE_CONTENT_URI.buildUpon()
                .appendPath(PATH_DIVES)
                .build();


        public static final String DIVE_TABLE_NAME = "diveTable";


        public static final String DIVE_TABLE_COLUMN_ID = "__diveId";
        public static final String DIVE_TABLE_COLUMN_DATE = "diveDate";
        public static final String DIVE_TABLE_COLUMN_AVERAGE_DEPTH = "diveAverageDepth";
        public static final String DIVE_TABLE_COLUMN_BOTTOM_TIME = "diveBottomTime";
        public static final String DIVE_TABLE_COLUMN_LESS_THAN_FIFTEEN_CENTIMETRES =
"diveLessThanFifteenCentimetres";
        public static final String DIVE_TABLE_COLUMN_FIFTEEN_TO_TWENTY_FIVE_CENTIMETRES =
"diveFifteenToTwentyFiveCentimetres";
        public static final String DIVE_TABLE_COLUMN_TWENTY_FIVE_TO_FORTY_CENTIMETRES =
"diveTwentyFiveToFortyCentimetres";
        public static final String DIVE_TABLE_COLUMN_GREATER_THAN_FORTY_CENTIMETRES =
"diveGreaterThanFortyCentimetres";
        public static final String DIVE_TABLE_COLUMN_SITE_ID = "__siteId";
        public static final String DIVE_TABLE_COLUMN_VOYAGE_ID = "__voyageId";


    }



    //    Inner class that defines the table contents of the RHIS table
    public static final class MantaEntry implements BaseColumns {


        //      The base CONTENT_URI used to query the Reefs from the Dive table from the content provider
        public static final Uri CONTENT_URI = BASE_CONTENT_URI.buildUpon()
                .appendPath(PATH_MANTAS)
                .build();


    //      For the moment, the Reefs query will simply run on the Dive table, extracting the Reef details
    //      from it. In future, the database structure will be updated and it will run on a dedicated Reefs
table.
    //      For the moment we keep references to all the Dive columns, but in future we will
    //      rationalise these to the actual entries in the Reef table.


        public static final String MANTA_TABLE_NAME = "mantaTable";
```

```
        public static final String MANTA_TABLE_COLUMN_ID = "__mantaId";
        public static final String MANTA_TABLE_COLUMN_DATE = "mantaDate";
        public static final String MANTA_TABLE_COLUMN_START_LAT = "mantaStartLatitude";
        public static final String MANTA_TABLE_COLUMN_START_LONG = "mantaStartLongitude";
        public static final String MANTA_TABLE_COLUMN_STOP_LAT = "mantaStopLatitude";
        public static final String MANTA_TABLE_COLUMN_STOP_LONG = "mantaStopLongitude";
        public static final String MANTA_TABLE_COLUMN_MEAN_LAT = "mantaMeanLatitude";
        public static final String MANTA_TABLE_COLUMN_MEAN_LONG = "mantaMeanLongitude";
        public static final String MANTA_TABLE_COLUMN_COTS = "mantaCOTS";
        public static final String MANTA_TABLE_COLUMN_SCARS = "mantaScars";
        public static final String MANTA_TABLE_COLUMN_SITE_ID = "__siteId";
        public static final String MANTA_TABLE_COLUMN_VOYAGE_ID = "__voyageId";

    }


    //    Inner class that defines the table contents of the RHIS table
    public static final class RhisEntry implements BaseColumns {

        //      The base CONTENT_URI used to query the Sites from the Dive table from the content provider
        public static final Uri CONTENT_URI = BASE_CONTENT_URI.buildUpon()
                .appendPath(PATH_RHIS)
                .build();

//      For the moment, the Rhis query will simply run on the Dive table, extracting the Rhis details
//      from it. In future, the database structure will be updated and it will run on a dedicated Rhis
table.
//      For the moment we keep references to all the Dive columns, but in future we will
//      rationalise these to the actual entries in the Rhis table.

        public static final String RHIS_TABLE_NAME = "rhisTable";

        public static final String RHIS_TABLE_COLUMN_ID = "__rhisId";
        public static final String RHIS_TABLE_COLUMN_DATE = "rhisDate";
        public static final String RHIS_TABLE_COLUMN_AVERAGE_CORAL_COVER = "rhisCoralCover";
        public static final String RHIS_TABLE_COLUMN_COTS_ADULTS = "rhisCOTSAdults";
        public static final String RHIS_TABLE_COLUMN_COTS_JUVENILES = "rhisCOTSJuveniles";
        public static final String RHIS_TABLE_COLUMN_VISIBILITY = "rhisVisibility";
        public static final String RHIS_TABLE_COLUMN_SITE_ID = "__siteId";

    }

}
```

## B.8 COTSDataProvider.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data;

import android.annotation.TargetApi;
```

```
import android.content.ContentProvider;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.net.Uri;
import androidx.annotation.NonNull;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.ReefEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.ReefPolygonsEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.SiteEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.SitePolygonsEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.VesselEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.VoyageEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.DiveEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.MantaEntry;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.RhisEntry;

//
// The COTSDataProvider class is where the main app cotsData.sqlite database is queried to load
// data into the custom Types of the app. It is important because much of the selection of the
// data that is displayed and processed occurs here, prior to analysis within the main class
// methods.
//
// The COTSDataProvider was established at a point where it was expected the COTS Control Centre
// would leverage extensive communication between multiple apps. Since then, better
// compartmentalisation between user tasks has reduced this requirement. As a result, although the
// framework for the COTSDataProvider is established, it is not currently leveraged efficiently,
// instead relying on multiple custom queries targeted at the immediate needs of the main COTS-DST
// app. In future, this will be restructures to provide a consistent approach to querying across
// all data types, with appropriate query builders etc.
//
// TODO: Restructure to have a consistent approach to querying, in terms of: 1) which arguments are
//  hardcoded into loaders; 2) which are accessed via "selection"; and 3) which are accessed via a
//  combination of "selection" and "selectionArgs"
//

public class COTSDataProvider extends ContentProvider {

    //
    // DEFINE DATAPROVIDER IDS
    // These are just human-readable integer variables that can be used in the switch statements
    // throughout this class. The only goal is to provide a unique integer for each type of
    // query we want to consider, so we space them out to give ourselves room to add additional
    // options if necessary as development continues
    //

    // REEFS - starting at 100
    public static final int CODE_REEFS_ALL = 100;
    public static final int CODE_REEFS_CONTROLLED = 101;
```

```
    // REEFPOLYGONS - starting at 150
    public static final int CODE_REEFPOLYGONS = 150;

    // SITES - starting at 200
    public static final int CODE_SITES_ALL = 200;
    public static final int CODE_SITES_WITH_SITENAME = 201;

    // SITEPOLYGONS - starting at 250
    public static final int CODE_SITEPOLYGONS = 250;

    // VESSELS - starting at 300
    public static final int CODE_VESSELS_ALL = 300;

    // VOYAGES - starting at 400
    public static final int CODE_VOYAGES_ALL = 400;

    // DIVES - starting at 500
    public static final int CODE_DIVES_ALL = 500;
    public static final int CODE_DIVES_AT_REEF = 501;
    public static final int CODE_DIVES_AT_SITE = 502;

    // MANTAS - starting at 600
    public static final int CODE_MANTAS_ALL = 600;
    public static final int CODE_MANTAS_AT_REEF = 601;

    //RHIS - starting at 700
    public static final int CODE_RHIS_ALL = 700;

    //
    // DEFINE PRIVATE CLASS VARIABLES
    //
    private static final UriMatcher sUriMatcher = buildUriMatcher();


    // This method provides a "UriMatcher" to determine which query to run based on the Uri used
    // to call the COTSDataProvider. That is, it converts a Uri like
    // "au.csiro.cotscontrolcentre_decisionsupporttool_0_0/reefs/" into one of the human-readable
    // DATAPROVIDER ID integers above. In practice, this isn't heavily leveraged yet, but it's a
    // necessary component of a DataProvider.
    public static UriMatcher buildUriMatcher() {

        /*
         * All paths added to the UriMatcher have a corresponding code to return when a match is
         * found. The code passed into the constructor of UriMatcher here represents the code to
         * return for the root URI. It's common to use NO_MATCH as the code for this case.
         */
        final UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH);
        final String authority = COTSDataContract.CONTENT_AUTHORITY;

//        Create a code for each type of URI.
```

```
        //
        // REEFS
        //

        // All Reefs: au.csiro.cotscontrolcentre_decisionsupporttool_0_0/reefs/
        matcher.addURI(
                authority,
                ReefEntry.CONTENT_URI
                        .buildUpon()
                        .build()
                        .getPath(),
                CODE_REEFS_ALL
        );

        // Controlled Reefs: au.csiro.cotscontrolcentre_decisionsupporttool_0_0/reefs/controlled/
        matcher.addURI(
                authority,
                ReefEntry.CONTENT_URI
                        .buildUpon()
                        .build()
                        .getPath() + "/controlled/",
                CODE_REEFS_CONTROLLED
        );



        //
        // REEF POLYGONS
        //

        // Reef Polygons from selected Reef:
au.csiro.cotscontrolcentre_decisionsupporttool_0_0/reefpolygons/
        matcher.addURI(
                authority,
                ReefPolygonsEntry.CONTENT_URI
                        .buildUpon()
                        .build()
                        .getPath(),
                CODE_REEFPOLYGONS
        );

        //
        // SITES
        //

        // All sites: au.csiro.cotscontrolcentre_decisionsupporttool_0_0/sites/
        matcher.addURI(
                authority,
                SiteEntry.CONTENT_URI
                        .buildUpon()
                        .build()
```

```
                        .getPath(),
                CODE_SITES_ALL
        );

        // All sites: au.csiro.cotscontrolcentre_decisionsupporttool_0_0/sites/where/sitename/*siteName*
        matcher.addURI(
                authority,
                SiteEntry.CONTENT_URI
                        .buildUpon()
                        .build()
                        .getPath()
                        + "/where/sitename/",
                CODE_SITES_WITH_SITENAME
        );

        //
        // SITEPOLYGONS
        //

        // SITE Polygons from selected sITE:
au.csiro.cotscontrolcentre_decisionsupporttool_0_0/sitepolygons/
        matcher.addURI(
                authority,
                SitePolygonsEntry.CONTENT_URI
                        .buildUpon()
                        .build()
                        .getPath(),
                CODE_SITEPOLYGONS
        );

        //
        // VESSELS
        //

        // All sites: au.csiro.cotscontrolcentre_decisionsupporttool_0_0/vessels/
        matcher.addURI(
                authority,
                VesselEntry.CONTENT_URI
                        .buildUpon()
                        .build()
                        .getPath(),
                CODE_VESSELS_ALL
        );

        //
        // VOYAGES
        //

        // All voyages: au.csiro.cotscontrolcentre_decisionsupporttool_0_0/voyages/
        matcher.addURI(
                authority,
```

```
                VoyageEntry.CONTENT_URI
                .buildUpon()
                .build()
                .getPath(),
                CODE_VOYAGES_ALL
        );

        //
        // DIVES
        //

        // All dives: au.csiro.cotscontrolcentre_decisionsupporttool_0_0/dives/
        matcher.addURI(
                authority,
                DiveEntry.CONTENT_URI
                        .buildUpon()
                        .build()
                        .getPath(),
                CODE_DIVES_ALL
        );

        // Dives at Reef: au.csiro.cotscontrolcentre_decisionsupporttool_0_0/dives/where/reef/*reefId*
        matcher.addURI(
                authority,
                DiveEntry.CONTENT_URI
                        .buildUpon()
                        .build()
                        .getPath()
                        + "/where/reef/",
                CODE_DIVES_AT_REEF
        );

        // Dives at Site:
au.csiro.cotscontrolcentre_decisionsupporttool_0_0/dives/where/sitename/*siteName*
        matcher.addURI(
                authority,
                DiveEntry.CONTENT_URI
                        .buildUpon()
                        .build()
                        .getPath()
                        + "/where/site/",
                CODE_DIVES_AT_SITE
        );



        //
        // MANTAS
        //

        // All Manta tows: au.csiro.cotscontrolcentre_decisionsupporttool_0_0/mantas/
        matcher.addURI(
```

```
                authority,
                MantaEntry.CONTENT_URI
                        .buildUpon()
                        .build()
                        .getPath(),
                CODE_MANTAS_ALL
        );

        // Mantas at Reef: au.csiro.cotscontrolcentre_decisionsupporttool_0_0/mantas/where/reef/*reefId*
        matcher.addURI(
                authority,
                MantaEntry.CONTENT_URI
                        .buildUpon()
                        .build()
                        .getPath()
                        + "/where/reef/",
                CODE_MANTAS_AT_REEF
        );

        //
        // RHIS
        //

        // All RHIS: au.csiro.cotscontrolcentre_decisionsupporttool_0_0/rhis/
        matcher.addURI(
                authority,
                RhisEntry.CONTENT_URI
                        .buildUpon()
                        .build()
                        .getPath(),
                CODE_RHIS_ALL
        );

        return matcher;
    }


    @Override
    public boolean onCreate() {
//      Note that onCreate is run on the main thread, so performing any lengthy operations will
//      cause app lag.

        return true;

    }


//    The query function is where we extract records from the database to populate the various
//    parts of our app. We open the database in read only mode when making a query. We use a
//    rawQuery rather than a query because some of our queries are too complicated for the Android
```

```
//    query function, and because I read that it is supposed to prevent SQL injection risks,
//    although I haven't verified that claim. In my opinion it's also easier for someone who knows
//    SQL to read and maintain.
//
//    In future this may be rewritten to make use of the modular nature of the Uri design so that
//    an explicit query does not need to be written out for every case, but for now we just do
//    what needs to be done
//
    @Override
    public Cursor query(@NonNull Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {

        COTSDataHelper COTSDataHelper = new COTSDataHelper(getContext());

        Cursor cursor;

        switch ( sUriMatcher.match(uri) ) {

            //
            // REEFS
            //

            case CODE_REEFS_ALL: {

                String query =
                        // We select all rows from REEFS_TABLE_NAME
                        "SELECT * FROM " + ReefEntry.REEF_TABLE_NAME;

                cursor = COTSDataHelper.getReadableDatabase().rawQuery(query,null);

                break;

            }

            case CODE_REEFS_CONTROLLED: {

                String query =
                        // We select all rows from REEFS_TABLE_NAME at which there are Dive and
                        // Manta records, through the use of the INNER JOIN function.
                        // TODO: Change this query to select Reefs with Manta OR Dive, not Manta
                        //  AND Dive
                        "SELECT DISTINCT " +
                         ReefEntry.REEF_TABLE_NAME + ".* " +
                        "FROM " +
                         SiteEntry.SITE_TABLE_NAME +
                         " INNER JOIN " + DiveEntry.DIVE_TABLE_NAME + " ON " + DiveEntry.DIVE_TABLE_NAME +
"." + DiveEntry.DIVE_TABLE_COLUMN_SITE_ID + "=" + SiteEntry.SITE_TABLE_NAME + "." +
SiteEntry.SITE_TABLE_COLUMN_ID +
                         " INNER JOIN " + ReefEntry.REEF_TABLE_NAME + " ON " + ReefEntry.REEF_TABLE_NAME +
"." + ReefEntry.REEF_TABLE_COLUMN_ID + "=" + SiteEntry.SITE_TABLE_NAME + "." +
SiteEntry.SITE_TABLE_COLUMN_REEF_ID;
```

```
                cursor = COTSDataHelper.getReadableDatabase().rawQuery(query,null);

                break;

            }

            //
            // REEFPOLYGONS
            //
            case CODE_REEFPOLYGONS: {

                String query;

                if ( selection.isEmpty() ) {

                    cursor = null;
                    break; /* We don't want to return a cursor for all Reefs, it would contain too many
items to be useful */

                }
                {

                    query = "SELECT * FROM " + ReefPolygonsEntry.REEF_POLYGONS_TABLE_NAME + " WHERE " +
selection;

                }

                cursor = COTSDataHelper.getReadableDatabase().rawQuery(query,null);

                break;

            }

            //
            // SITES
            //

            case CODE_SITES_ALL: {

                String query;

                String basequery =
                        "SELECT DISTINCT " +
                        SiteEntry.SITE_TABLE_NAME + ".*" +
                        " FROM " + SiteEntry.SITE_TABLE_NAME +
                        " LEFT JOIN " + DiveEntry.DIVE_TABLE_NAME + " ON " + DiveEntry.DIVE_TABLE_NAME +
"." + DiveEntry.DIVE_TABLE_COLUMN_SITE_ID + " = " + SiteEntry.SITE_TABLE_NAME +"." +
SiteEntry.SITE_TABLE_COLUMN_ID +
                        " LEFT JOIN " + MantaEntry.MANTA_TABLE_NAME + " ON " + MantaEntry.MANTA_TABLE_NAME
+ "." + MantaEntry.MANTA_TABLE_COLUMN_SITE_ID + " = " + SiteEntry.SITE_TABLE_NAME +"." +
SiteEntry.SITE_TABLE_COLUMN_ID;
```

```
            String groupby = " GROUP BY " + SiteEntry.SITE_TABLE_NAME +"." +
SiteEntry.SITE_TABLE_COLUMN_ID;

            if ( selection.isEmpty() ) {

                query = basequery + groupby;

            } else {

                query = basequery + " WHERE " + selection + groupby;

            }

            cursor = COTSDataHelper.getReadableDatabase().rawQuery(query,null);

            break;

        }

        case CODE_SITES_WITH_SITENAME: {

            String query;

            String basequery = "SELECT " +
                    SiteEntry.SITE_TABLE_NAME +
                    " FROM " + SiteEntry.SITE_TABLE_NAME;

            if ( selection.isEmpty() ) {

                query = basequery;

            } else {

                query = basequery + " WHERE " + selection;

            }

            cursor = COTSDataHelper.getReadableDatabase().rawQuery(query,null);

            break;

        }

        //
        // VESSELS
        //

        case CODE_VESSELS_ALL: {

            String query =
                    // We select all rows from SITES_TABLE_NAME
```

```
                        "SELECT * FROM " + VesselEntry.VESSEL_TABLE_NAME;

                cursor = COTSDataHelper.getReadableDatabase().rawQuery(query,null);

                break;
            }

            //
            // SITEPOLYGONS
            //

            case CODE_SITEPOLYGONS: {

                String query;

                if ( selection.isEmpty() ) {

                    cursor = null;
                    break; /* We don't want to return a cursor for all Sites, it would contain too many
items to be useful */

                }
                {
                    query =
                            "SELECT DISTINCT " +
                            SitePolygonsEntry.SITE_POLYGONS_TABLE_NAME + ".* " +
                            "FROM " +
                            SitePolygonsEntry.SITE_POLYGONS_TABLE_NAME +
                            " LEFT JOIN " + SiteEntry.SITE_TABLE_NAME + " ON " + SiteEntry.SITE_TABLE_NAME
+ "." + SiteEntry.SITE_TABLE_COLUMN_ID + "=" + SitePolygonsEntry.SITE_POLYGONS_TABLE_NAME + "." +
SitePolygonsEntry.SITE_POLYGONS_TABLE_COLUMN_SITE_ID +
                            " WHERE " + selection +
                            " ORDER BY " + SitePolygonsEntry.SITE_POLYGONS_TABLE_NAME + "." +
SitePolygonsEntry.SITE_POLYGONS_TABLE_COLUMN_SITE_ID + ", " + SitePolygonsEntry.SITE_POLYGONS_TABLE_NAME +
"." + SitePolygonsEntry.SITE_POLYGONS_TABLE_COLUMN_POINT_ORDER;
                }

                cursor = COTSDataHelper.getReadableDatabase().rawQuery(query,null);

                break;

            }

            //
            // VOYAGES SITES
            //

            case CODE_VOYAGES_ALL: {

                String query;

                String basequery = "SELECT DISTINCT " +
```

```
                        VoyageEntry.VOYAGE_TABLE_NAME + ".*" +
                        " FROM " + VoyageEntry.VOYAGE_TABLE_NAME +
                        " LEFT JOIN " + DiveEntry.DIVE_TABLE_NAME + " ON " + DiveEntry.DIVE_TABLE_NAME +
"." + DiveEntry.DIVE_TABLE_COLUMN_VOYAGE_ID + " = " + VoyageEntry.VOYAGE_TABLE_NAME +"." +
VoyageEntry.VOYAGE_TABLE_COLUMN_ID +
                        " LEFT JOIN " + MantaEntry.MANTA_TABLE_NAME + " ON " + MantaEntry.MANTA_TABLE_NAME
+ "." + MantaEntry.MANTA_TABLE_COLUMN_VOYAGE_ID + " = " + VoyageEntry.VOYAGE_TABLE_NAME +"." +
VoyageEntry.VOYAGE_TABLE_COLUMN_ID +
                        " LEFT JOIN " + SiteEntry.SITE_TABLE_NAME + " ON " + SiteEntry.SITE_TABLE_NAME +
"." + SiteEntry.SITE_TABLE_COLUMN_ID + " = " + DiveEntry.DIVE_TABLE_NAME + "." +
DiveEntry.DIVE_TABLE_COLUMN_SITE_ID;


                String groupby = " GROUP BY " + VoyageEntry.VOYAGE_TABLE_NAME +"." +
VoyageEntry.VOYAGE_TABLE_COLUMN_ID;


                if ( selection == null ) {


                    query = basequery + groupby;


                } else {


                    query = basequery + " WHERE " + selection + groupby;


                }


                cursor = COTSDataHelper.getReadableDatabase().rawQuery(query,null);


                break;


            }


            //
            // DIVES
            //


            case CODE_DIVES_ALL: {


                String query;


                if ( selection.isEmpty() ) {
                    query = "SELECT * FROM " + DiveEntry.DIVE_TABLE_NAME;
                }
                {
                    query = "SELECT * FROM " + DiveEntry.DIVE_TABLE_NAME + " WHERE " + selection;
                }


                cursor = COTSDataHelper.getReadableDatabase().rawQuery(query,null);


                break;


            }


            case CODE_DIVES_AT_REEF: {
```

```
                //TODO: Change this query to get all Dives after the most recent Voyage with Manta tow
                String query =
                        "SELECT DISTINCT " +
                        DiveEntry.DIVE_TABLE_NAME + ".* " +
                        "FROM " +
                        DiveEntry.DIVE_TABLE_NAME +
                        " LEFT JOIN " + SiteEntry.SITE_TABLE_NAME + " ON " + SiteEntry.SITE_TABLE_NAME +
"." + SiteEntry.SITE_TABLE_COLUMN_ID + "=" + DiveEntry.DIVE_TABLE_NAME + "." +
DiveEntry.DIVE_TABLE_COLUMN_SITE_ID +
                        " WHERE " + selection;

                cursor = COTSDataHelper.getReadableDatabase().rawQuery(query,null);

                break;

            }

            case CODE_DIVES_AT_SITE: {

                String query =
                        "SELECT " + DiveEntry.DIVE_TABLE_NAME + ".* FROM " +
                                DiveEntry.DIVE_TABLE_NAME +
                                " WHERE " + selection;

                cursor = COTSDataHelper.getReadableDatabase().rawQuery(query,null);

                break;

            }

            //
            // MANTAS
            //

            case CODE_MANTAS_ALL: {

                String query =
                        // We select all rows from MANTAS_TABLE_NAME
                        "SELECT * FROM " + MantaEntry.MANTA_TABLE_NAME;

                cursor = COTSDataHelper.getReadableDatabase().rawQuery(query,null);

                break;
            }

            case CODE_MANTAS_AT_REEF: {

                String query =
                        "SELECT DISTINCT " + MantaEntry.MANTA_TABLE_NAME + ".* FROM " +
                                MantaEntry.MANTA_TABLE_NAME +
```

```
                                " INNER JOIN " + SiteEntry.SITE_TABLE_NAME + " ON " +
SiteEntry.SITE_TABLE_NAME + "." + SiteEntry.SITE_TABLE_COLUMN_ID + "=" + MantaEntry.MANTA_TABLE_NAME + "."
+ MantaEntry.MANTA_TABLE_COLUMN_SITE_ID +
                                " WHERE " + selection;


                cursor = COTSDataHelper.getReadableDatabase().rawQuery(query,null);


                break;


            }


            //
            // RHIS
            //


            case CODE_RHIS_ALL: {


                String query =
                        "SELECT * FROM " + RhisEntry.RHIS_TABLE_NAME;


                cursor = COTSDataHelper.getReadableDatabase().rawQuery(query, null);


                break;


            }


            default:
                throw new UnsupportedOperationException("Unknown uri: " + uri);
        }


        cursor.setNotificationUri(getContext().getContentResolver(), uri);


        return cursor;
    }


    //
    // COMPULSORY METHODS
    //


    // These are not yet used in the current implementation, but may be developed further as the
    // COTSContentProvider is leveraged more completely.


    @Override
    public int delete(@NonNull Uri uri, String selection, String[] selectionArgs) {
        throw new RuntimeException(
                "Not yet implemented");
    }


    @Override
    public String getType(@NonNull Uri uri) {
        throw new RuntimeException("Not yet implemented");
```

```
    }

    @Override
    public Uri insert(@NonNull Uri uri, ContentValues values) {
        throw new RuntimeException(
                "Not yet implemented");
    }

    @Override
    public int update(@NonNull Uri uri, ContentValues values, String selection, String[] selectionArgs) {
        throw new RuntimeException("Not yet implemented");
    }

//     This is an internal method to assist the testing framework in running smoothly, described here:
//     http://developer.android.com/reference/android/content/ContentProvider.html#shutdown()
    @Override
    @TargetApi(11)
    public void shutdown() {
        COTSDataHelper COTSDataHelper = new COTSDataHelper(getContext());
        COTSDataHelper.close();
        super.shutdown();
    }

}
```

## B.9 COTSDataHelper.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

/**
 * Created by fle125 on 27/03/2017.
 */

public class COTSDataHelper extends SQLiteOpenHelper {

    // We need to do some jiggery-pokery here to load the pre-existing cotsData.sqlite database into the
app.
    // I'm not sure why - maybe the system needs to be register it as a data source or something - but I
```

```
    // tried just copying it to the correct directory of the Android file system itself and it didn't work.
    // All the advice online suggests this is the best way to do it, and it has the benefit that I can
update
    // the database locally before uploading the APK with it compiled in.

    // The instance, in order to make the COTSDataHelper a singleton and avoid unclosed databases and
memory leaks
//    private static COTSDataHelper sInstance;

    // The database path
    private static String DATABASE_PATH;

    // The database name
    private static final String DATABASE_NAME = "cotsData.sqlite";

    // If you change the database schema, you must increment the database version so that the onUpdate
    // function is called on next load.
    private static final int DATABASE_VERSION = 1;

    // Some private variables for what follows:

    private SQLiteDatabase mDatabase;

    private final Context mContext;

//
//    The following three functions are the compulsory functions that need to be overridden
//

//    public static synchronized COTSDataHelper getInstance(Context context) {
//
//        // Use the application context, which will ensure that you
//        // don't accidentally leak an Activity's context.
//        // See this article for more information: http://bit.ly/6LRzfx
//        if (sInstance == null) {
//            sInstance = new COTSDataHelper(context.getApplicationContext());
//        }
//        return sInstance;
//    }

    // Make constructor private to calculate the DATABASE_PATH in the public version below before calling
super

    public COTSDataHelper(Context context ) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
        DATABASE_PATH = "/data/data/" + context.getPackageName() + "/" + "databases/";
        this.mContext = context;
        boolean dbexist = checkDatabase();
        if(dbexist)
        {
```

```
            openDatabase();
        }
        else
        {
            createDatabase();
        }
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {
    }

    /**
     * Creates a empty database on the system and rewrites it with your own database.
     * */
    public void createDatabase() {

        boolean dbExist = checkDatabase();

        if(dbExist){
            //do nothing - database already exist
        }else{

            //By calling this method and empty database will be created into the default system path
            //of your application so we are gonna be able to overwrite that database with our database.
            this.getReadableDatabase();

            try {

                copyDatabase();

            } catch (IOException e) {

                e.printStackTrace();

            }
        }

    }

//
//  The following functions are informed by knowledgeable internet sources about loading pre-existing
//  data into an app
//

//  Check if the database already exist to avoid re-copying the file each time you open the application.
```

```
//  @return true if it exists, false if it doesn't
    private boolean checkDatabase(){

        boolean checkdb = false;
        SQLiteDatabase checkDB = null;

        try{
            String myPath = DATABASE_PATH + DATABASE_NAME;
            File dbfile = new File(myPath);
            checkDB = SQLiteDatabase.openDatabase(myPath, null, SQLiteDatabase.OPEN_READONLY);
            checkdb = dbfile.exists();

        }catch(SQLiteException e){

            //database does't exist yet.

        }

        if(checkDB != null){

            checkDB.close();

        }

//        return checkDB != null ? true : false;
        return checkdb;
    }

//  Copies your database from your local assets-folder to the just created empty database in the
//  system folder, from where it can be accessed and handled. This is done by transferring byte
//  streams.
    private void copyDatabase() throws IOException {

        //Open your local db as the input stream
        InputStream myInput = mContext.getAssets().open(DATABASE_NAME);

        // Path to the just created empty db
        String outFileName = DATABASE_PATH + DATABASE_NAME;

        //Open the empty db as the output stream
        OutputStream myOutput = new FileOutputStream(outFileName);

        //transfer bytes from the inputfile to the outputfile
        byte[] buffer = new byte[1024];
        int length;
        while ((length = myInput.read(buffer))>0){
            myOutput.write(buffer, 0, length);
        }

        //Close the streams
```

```
        myOutput.flush();
        myOutput.close();
        myInput.close();

    }

    public SQLiteDatabase openDatabase() throws android.database.sqlite.SQLiteException{

        //Open the database
        String myPath = DATABASE_PATH + DATABASE_NAME;
        mDatabase = SQLiteDatabase.openDatabase(myPath, null, SQLiteDatabase.OPEN_READWRITE);
        return mDatabase;

    }

    @Override
    public synchronized void close() {

        if(mDatabase != null) {
            mDatabase.close();
        }

        super.close();

    }

}
```

## B.10 Dive.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types;

import android.database.Cursor;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Comparator;
import java.util.Date;
import java.util.concurrent.TimeUnit;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.MainActivity;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.DiveEntry;

/**
 * Created by fle125 on 31/03/2017.
 */

public class Dive implements Comparable<Dive> {
```

```
    // Private Variables
    // These are the components intrinsic to a Dive, rather than values that should be derived
    // from the Voyage on which the Dive occurred, or during a nearby Rhis, for instance.
    // For instance - the Dive date and the number of CoTS removed are intrinsic to a specific dive.
    // On the other hand, the vessel that the Dive occurred on is determined by the Voyage of which
    // it was a part. Similarly, the average coral cover is determined by the Rhis that is assigned
    // to this Dive. We do not want to record the vessel or the average coral cover as part of the
    // Dive, but we do want to provide the underlying structure across the various classes that
    // could return them, given a Dive, with its diveId , siteid and voyageId, a list of Voyages,
    // each with its vesselId, and a list of Rhis that occurred at siteIds.

    private int _diveId;
    private String _diveDate;
    private double _diveAverageDepth;
    private int _diveBottomTime;
    private int _diveLessThanFifteenCentimetres;
    private int _diveFifteenToTwentyFiveCentimetres;
    private int _diveTwentyFiveToFortyCentimetres;
    private int _diveGreaterThanFortyCentimetres;
    private int _siteId;
    private int _voyageId;

    //
    // CONSTRUCTORS
    //

    // Empty constructor
    public Dive(){
    }

    // Component constructor
    public Dive(int diveId, String diveDate, double diveAverageDepth, int diveBottomTime, int
diveLessThanFifteenCentimetres, int diveFifteenToTwentyFiveCentimetres, int
diveTwentyFiveToFortyCentimetres, int diveGreaterThanFortyCentimetres, int siteId, int voyageId){
        this._diveId = diveId;
        this._diveDate = diveDate;
        this._diveAverageDepth = diveAverageDepth;
        this._diveBottomTime = diveBottomTime;
        this._diveLessThanFifteenCentimetres = diveLessThanFifteenCentimetres;
        this._diveFifteenToTwentyFiveCentimetres = diveFifteenToTwentyFiveCentimetres;
        this._diveTwentyFiveToFortyCentimetres = diveTwentyFiveToFortyCentimetres;
        this._diveGreaterThanFortyCentimetres = diveGreaterThanFortyCentimetres;
        this._siteId = siteId;
        this._voyageId = voyageId;
    }

    // Cursor constructor
    public Dive(Cursor cursor) {
        this._diveId = cursor.getInt(cursor.getColumnIndex(DiveEntry.DIVE_TABLE_COLUMN_ID));
        this._diveDate = cursor.getString(cursor.getColumnIndex(DiveEntry.DIVE_TABLE_COLUMN_DATE));
```

```
        this._diveAverageDepth =
cursor.getDouble(cursor.getColumnIndex(DiveEntry.DIVE_TABLE_COLUMN_AVERAGE_DEPTH));
        this._diveBottomTime =
cursor.getInt(cursor.getColumnIndex(DiveEntry.DIVE_TABLE_COLUMN_BOTTOM_TIME));
        this._diveLessThanFifteenCentimetres =
cursor.getInt(cursor.getColumnIndex(DiveEntry.DIVE_TABLE_COLUMN_LESS_THAN_FIFTEEN_CENTIMETRES));
        this._diveFifteenToTwentyFiveCentimetres =
cursor.getInt(cursor.getColumnIndex(DiveEntry.DIVE_TABLE_COLUMN_FIFTEEN_TO_TWENTY_FIVE_CENTIMETRES));
        this._diveTwentyFiveToFortyCentimetres =
cursor.getInt(cursor.getColumnIndex(DiveEntry.DIVE_TABLE_COLUMN_TWENTY_FIVE_TO_FORTY_CENTIMETRES));
        this._diveGreaterThanFortyCentimetres =
cursor.getInt(cursor.getColumnIndex(DiveEntry.DIVE_TABLE_COLUMN_GREATER_THAN_FORTY_CENTIMETRES));
        this._siteId = cursor.getInt(cursor.getColumnIndex(DiveEntry.DIVE_TABLE_COLUMN_SITE_ID));
        this._voyageId = cursor.getInt(cursor.getColumnIndex(DiveEntry.DIVE_TABLE_COLUMN_VOYAGE_ID));
    }


    //
    // GETTERS
    //


    // We provide individual public getter functions so that pieces of data can be read from
    // each Dive object.


    public int getDiveId(){
        return this._diveId;
    }


    public String getDiveDate(){
        return this._diveDate;
    }


    // At the moment, all dates are stored as YYYY-MM-DD strings, based on the structure available
    // to us in the GBRMPA Eye-On-The-Reef database exports. In future, this may be refined to
    // store date as UNIX time instead, to facilitate better range searches. For now, because we
    // often need to access the diveDate as a Date object for comparison or calculation, we provide
    // a method that: 1) checks that the diveDate is not null, and if it's OK, returns the diveDate
    // as a Date object.
    public Date getDiveDateAsDate() {


        SimpleDateFormat sdfDiveDate = new SimpleDateFormat( "yyyy-MM-dd");
        Date returnDate = null;


        try {


            returnDate = sdfDiveDate.parse( this._diveDate );


        } catch (ParseException e) {


            e.printStackTrace();


        }


        return returnDate;
```

```
}

// Getting averageDepth
public double getDiveAverageDepth(){
    return this._diveAverageDepth;
}

// Getting bottomTime
public int getBottomTime(){
    return this._diveBottomTime;
}

// Getting lessThanFifteenCentimetres
public int getLessThanFifteenCentimetres(){
    return this._diveLessThanFifteenCentimetres;
}

// Getting fifteenToTwentyFiveCentimetres
public int getFifteenToTwentyFiveCentimetres() { return this._diveFifteenToTwentyFiveCentimetres; }

// Getting twentyFiveToFortyCentimetres
public int getTwentyFiveToFortyCentimetres(){
    return this._diveTwentyFiveToFortyCentimetres;
}

// Getting greaterThanFortyCentimetres
public int getGreaterThanFortyCentimetres(){
    return this._diveGreaterThanFortyCentimetres;
}

// Getting siteId
public int getSiteId(){ return this._siteId; }

// Getting siteId
public int getVoyageId(){ return this._voyageId; }


//
// DERIVED VALUES
//

// We also provide functions for derived values, like the total number of CoTS removed
// and the catch per unit effort

// Total number of CoTS removed
public int totalCoTS(){

    int totalCoTS =
            this._diveLessThanFifteenCentimetres +
```

```
            this._diveFifteenToTwentyFiveCentimetres +
            this._diveTwentyFiveToFortyCentimetres +
            this._diveGreaterThanFortyCentimetres;

        return totalCoTS;
    }

    public double cpue(){

        double totalCoTS = this.totalCoTS();
        double bottomTime = this._diveBottomTime;
        double cpue = totalCoTS / bottomTime;
        return cpue;

    }

    //
    // This derived function estimates how much coral the COTS removed would consume each day,
    // in centimetres-squared, had they not been removed, based on their size class. The data is
    // approximate, and comes from fitting a quadratic damage function vs COTS size to the
    // data in Figure 4 of Keesing and Lucas (1992)
    public double avoidedDamage(){

        return (
                22 * this.getLessThanFifteenCentimetres() +
                57 * this.getFifteenToTwentyFiveCentimetres() +
                149 * this.getTwentyFiveToFortyCentimetres() +
                348 * this.getGreaterThanFortyCentimetres()
                ) / 10000d;
    }


    public long numberOfDaysSinceDive(){

        Date diveDate = this.getDiveDateAsDate();

        Calendar todaysDate = Calendar.getInstance();
        long timeSinceDive = todaysDate.getTime().getTime() - diveDate.getTime();
        long numberOfDaysSinceDive = TimeUnit.MILLISECONDS.toDays( timeSinceDive );

        return numberOfDaysSinceDive;

    }

    public boolean belowEcologicalThreshold(){

        return ( this.cpue() <= MainActivity.ecologicalThresholdCPUE );

    }
```

```
//
// COMPARATORS
//

@Override
public int compareTo(Dive dive) {

    Date thisDiveDate = null;
    Date diveDiveDate = null;
    SimpleDateFormat sdfDiveDate = new SimpleDateFormat( "yyyy-MM-dd");

    try {

        thisDiveDate = sdfDiveDate.parse(this._diveDate);
        diveDiveDate = sdfDiveDate.parse(dive.getDiveDate());

    } catch (Exception e) {

        e.printStackTrace();

    }


    if ( diveDiveDate.after( thisDiveDate ) )

        return -1;

    else if ( diveDiveDate.before( thisDiveDate ) )

        return 1;

    else

        return 0;

}

public static Comparator<Dive> getDiveDateComparator() {

    return new Comparator<Dive>() {

        public int compare(Dive dive1, Dive dive2) {

            Date dive1Date = null;
            Date dive2Date = null;
            SimpleDateFormat sdfDiveDate = new SimpleDateFormat("yyyy-MM-dd");

            try {
```

```
                dive1Date = sdfDiveDate.parse( dive1.getDiveDate() );
                dive2Date = sdfDiveDate.parse( dive2.getDiveDate() );

            } catch (Exception e) {

                e.printStackTrace();

            }


            if ( dive1Date.after( dive2Date ) )

                return -1;

            else if ( dive1Date.before( dive2Date ) )

                return 1;

            else

                return 0;

        }

    };

}


public static Comparator<Dive> getDiveCullNumberComparator() {

    return new Comparator<Dive>() {

        public int compare(Dive dive1, Dive dive2) {

            if ( dive1.totalCoTS() > dive2.totalCoTS() )

                return -1;

            else if ( dive1.totalCoTS() < dive2.totalCoTS() )

                return 1;

            else

                return 0;

        }

    };
```

```
    }

}
```

## B.11 DiveList.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists;

import android.util.Log;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Dive;

//
// The purpose of these custom TypeList types, encompassing each of our custom Types, is to provide
// functionality related to the physical characteristics of the custom Types. For example, it
// makes sense to be able to call a method that returns the most recent Dives performed at each
// Site in a list of Dives.
//
// There is an additional major benefit, in that we can store a HashMap of the indices of the
// TypeList when it is created. This allows us to write functions that can quickly find at item
// either by its index or by its Type identifier number. In the original data tables these two
// numbers are the same: i.e. the 131st Site in the database siteTable has a SiteId of 131.
// However, in a generic SiteList this is not the case - if we make a list of Sites at a Reef, the
// Site with SiteId 131 might be the 10th item in that particular list.
//
// Implementation note: It is not clear to me whether it is more efficient to create a HashMap of
// < typeId, _typeListIndex >, as I do in this code, and then use that to look up the position of an
// item with a given typeId in the typeList, or whether I should make a HashMap <typeId, Type Object>
// directly. This way works for now, but we could do some performance testing later.
//
// Given the commonality between these TypeList classes, I should make a BaseTypeList class and
// then only include the code specific to each Type within the TypeClass, but I haven't got
// around to this simplification yet.
//
// TODO: Change implementation of common elements of TypeList classes via a BaseTypeClass

public class DiveList implements
        Iterable<Dive> {
```

```
private List<Dive> _diveList = new ArrayList<Dive>();
private HashMap<Integer, Integer> _diveListIndex = new HashMap< Integer, Integer >();

//
// CONSTRUCTORS
//

// Empty constructor
public DiveList(){
}

// Constructor
public DiveList(List<Dive> diveList ){

    this._diveList = diveList;

    for ( int i = 0; i < diveList.size(); i++ ) {

        this._diveListIndex.put( diveList.get(i).getDiveId(), i );

    }

}

//
// SETTERS
//

public void add( Dive dive ) {

    this._diveList.add( dive );

    this._diveListIndex.put( dive.getDiveId(), this._diveList.size() - 1 );

}

public void clear() {

    this._diveList.clear();

    this._diveListIndex.clear();

}

//
// GETTERS
//

public Dive get( int i ) {
```

```
    return this._diveList.get( i );

}

// Get Dive from diveList by diveId
public Dive getDiveByDiveId( int diveId ){

    int diveTableIndex  = this._diveListIndex.get( diveId );

    return _diveList.get( diveTableIndex );

}

// Get a list of Dives from diveList by a list of diveIds
public DiveList getDivesByDiveIds( List<Integer> diveIds ){

    DiveList returnDives = new DiveList();

    for ( int diveId : diveIds ) {

        int diveIndex = this._diveListIndex.get( diveId );

        returnDives.add( this._diveList.get( diveIndex ) );

    }

    return returnDives;

}

public DiveList getDivesOnMostRecentVoyage(){

    if ( this._diveList.isEmpty() ) {

        return ( new DiveList ( new ArrayList<Dive>() ) );

    } else {

        // Get a copy of the underlying list of Dives, so we don't sort the actual DiveList
        List<Dive> diveListCopy = new ArrayList<>(this._diveList);

        // Sort the copy
        Collections.sort(diveListCopy, Dive.getDiveDateComparator());

        // Find the voyageId of the most recent Dive
        int voyageId = diveListCopy.get(0).getVoyageId();

        // Return a list of the Dive with that voyageId
        return this.getDivesByVoyageId( voyageId );
```

```
        }

    }

    // Get copy of DiveList
    public List<Dive> getDiveListCopy(){

        // Only ever return a copy of the _mantaList so we don't muck up the ordering of the
        // <key,value> pairs in _mantaListIndex

        List<Dive> diveListCopy = new ArrayList<Dive>( this._diveList );

        return diveListCopy;

    }

    public DiveList getDivesByVoyageId( int voyageId ){

        DiveList returnDiveList = new DiveList();

        for ( Dive dive : this._diveList ){

            if ( dive.getVoyageId() == voyageId ){

                returnDiveList.add( dive );

            }

        }

        return returnDiveList;

    }

    public DiveList getDivesBySiteId( int siteId ){

        DiveList returnDiveList = new DiveList();

        for ( Dive dive : this._diveList ){

            if ( dive.getSiteId() == siteId ){

                returnDiveList.add( dive );

            }

        }

        return returnDiveList;
```

```
    }

    //
    // DERIVED VALUES
    //

    // We also provide functions for derived values, like the total number of CoTS removed
    // and the catch per unit effort achieved across a list of Type

    public Integer getTotalCOTS(){

        if ( this._diveList.isEmpty() ) {

            return 0;

        } else {

            int totalCOTS = 0;

            for ( Dive dive : this._diveList ) {

                totalCOTS += dive.totalCoTS();

            }

            return totalCOTS;

        }

    }

    public boolean isEmpty() {

        return this._diveList.isEmpty();

    }


    @Nullable
    public Date getMostRecentDiveDate() {

        if ( this._diveList.isEmpty() ) {

            return null;

//            return ( new DiveList ( new ArrayList<Dive>() ) );

        } else {

            return getMostRecentDive().getDiveDateAsDate();
```

```
        }

    }

    @Nullable
    public Integer getMostRecentDiveVoyageId() {

        if ( this._diveList.isEmpty() ) {

            return null;

//            return ( new DiveList ( new ArrayList<Dive>() ) );

        } else {

            return getMostRecentDive().getVoyageId();

        }

    }

    @Nullable
    public Dive getMostRecentDive() {

        if ( this._diveList.isEmpty() ) {

            return null;

//            return ( new DiveList ( new ArrayList<Dive>() ) );

        } else {

            // Get a copy of the underlying list of Dives, so we don't sort the actual DiveList
            List<Dive> diveListCopy = new ArrayList<>( this._diveList );

            // Sort the copy
            Collections.sort( diveListCopy, Dive.getDiveDateComparator() );

            // Find the voyageId of the most recent Dive
            int voyageId = diveListCopy.get( 0 ).getVoyageId();

            // Return a list of the Dive with that voyageId
            return this.getDivesByVoyageId( voyageId ).get( 0 );

        }

    }

    // Get Dive from diveList by diveId
```

```
public int getDiveListIndexByDiveId( int diveId ){

    return this._diveListIndex.get( diveId );

}

//
// Iterators
//

@NonNull
@Override
public Iterator<Dive> iterator() {

    return this._diveList.iterator();

}

// This function should only be used on a DiveList describing Dives conducted at the same
// Site on the same Voyage. At the moment we just post a Log message if this criterion is not
// met - in future we should Throw and exception.
// TODO: Update createCompositeDive to throw exception on inappropriate input Dives
public Dive createCompositeDive (){

    int diveId;
    String diveDate;
    double diveAverageDepth = 0;
    int diveBottomTime = 0;
    int diveLessThanFifteenCentimetres = 0;
    int diveFifteenToTwentyFiveCentimetres = 0;
    int diveTwentyFiveToFortyCentimetres = 0;
    int diveGreaterThanFortyCentimetres = 0;
    int siteId;
    int voyageId;

    if ( !this._diveList.isEmpty() ) {

        diveDate = this._diveList.get(0).getDiveDate();
        siteId = this._diveList.get(0).getSiteId();
        voyageId = this._diveList.get(0).getVoyageId();

        for (Dive dive : this._diveList) {

            diveAverageDepth += dive.getDiveAverageDepth();
            diveBottomTime += dive.getBottomTime();
            diveLessThanFifteenCentimetres += dive.getLessThanFifteenCentimetres();
            diveFifteenToTwentyFiveCentimetres += dive.getFifteenToTwentyFiveCentimetres();
            diveTwentyFiveToFortyCentimetres += dive.getTwentyFiveToFortyCentimetres();
            diveGreaterThanFortyCentimetres += dive.getGreaterThanFortyCentimetres();
```

```
                if ((dive.getSiteId() != siteId) || (dive.getVoyageId() != voyageId)) {

                    Log.d("CCC_DIVE", "Dives should not have been combined because they span multiple Sites
or Voyages");

                }

            }

            diveAverageDepth = diveAverageDepth / this._diveList.size();

            return new Dive(-1, diveDate, diveAverageDepth, diveBottomTime, diveLessThanFifteenCentimetres,
diveFifteenToTwentyFiveCentimetres, diveTwentyFiveToFortyCentimetres, diveGreaterThanFortyCentimetres,
siteId, voyageId);

        } else {

            return new Dive();

        }

    }

}
```

## B.12 Manta.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types;

import android.database.Cursor;

import com.google.android.gms.maps.model.LatLng;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Comparator;
import java.util.Date;
import java.util.concurrent.TimeUnit;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.MainActivity;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.MantaEntry;

/**
 * Created by fle125 on 31/03/2017.
 */

public class Manta implements Comparable<Manta> {

    // Private Variables
    // These are the components intrinsic to a Manta, rather than values that should be derived
```

```
    // from the Site at which the Manta occurred or the Voyage that the Manta occurred on.
    // For instance - the Manta Start and Stop latitude and longitude are intrinsic to a Manta.
    // On the other hand, the name of the Reef at which the Manta occurred is a function of the Site
    // near which the Manta occurred.

    private int _mantaId;
    private String _mantaDate;
    private double _mantaStartLat;
    private double _mantaStartLong;
    private double _mantaStopLat;
    private double _mantaStopLong;
    private double _mantaMeanLat;
    private double _mantaMeanLong;
    private int _mantaCots;
    private String _mantaScars;
    private int _siteId;
    private int _voyageId;

    //
    // CONSTRUCTORS
    //

    // Empty constructor
    public Manta(){
    }

    // Constructor based on passing all required values
    public Manta(int mantaId, String mantaDate, double mantaStartLat, double mantaStartLong, double
mantaStopLat, double mantaStopLong, double mantaMeanLat, double mantaMeanLong, int mantaCots, String
mantaScars, Integer siteId, Integer voyageId ){

        this._mantaId = mantaId;
        this._mantaDate = mantaDate;
        this._mantaStartLat = mantaStartLat;
        this._mantaStartLong = mantaStartLong;
        this._mantaStopLat = mantaStopLat;
        this._mantaStopLong = mantaStopLong;
        this._mantaMeanLat = mantaMeanLat;
        this._mantaMeanLong = mantaMeanLong;
        this._mantaCots = mantaCots;
        this._mantaScars = mantaScars;
        this._siteId = siteId;
        this._voyageId = voyageId;

    }

    // Constructor based on passing a single Cursor to a SiteEntry
    public Manta(Cursor cursor) {

        this._mantaId = cursor.getInt(cursor.getColumnIndex(MantaEntry.MANTA_TABLE_COLUMN_ID));
        this._mantaDate = cursor.getString(cursor.getColumnIndex(MantaEntry.MANTA_TABLE_COLUMN_DATE));
```

```
        this._mantaStartLat =
cursor.getDouble(cursor.getColumnIndex(MantaEntry.MANTA_TABLE_COLUMN_START_LAT));
        this._mantaStartLong =
cursor.getDouble(cursor.getColumnIndex(MantaEntry.MANTA_TABLE_COLUMN_START_LONG));
        this._mantaStopLat =
cursor.getDouble(cursor.getColumnIndex(MantaEntry.MANTA_TABLE_COLUMN_STOP_LAT));
        this._mantaStopLong =
cursor.getDouble(cursor.getColumnIndex(MantaEntry.MANTA_TABLE_COLUMN_STOP_LONG));
        this._mantaMeanLat =
cursor.getDouble(cursor.getColumnIndex(MantaEntry.MANTA_TABLE_COLUMN_MEAN_LAT));
        this._mantaMeanLong =
cursor.getDouble(cursor.getColumnIndex(MantaEntry.MANTA_TABLE_COLUMN_MEAN_LONG));
        this._mantaCots = cursor.getInt(cursor.getColumnIndex(MantaEntry.MANTA_TABLE_COLUMN_COTS));
        this._mantaScars = cursor.getString(cursor.getColumnIndex(MantaEntry.MANTA_TABLE_COLUMN_SCARS));
        this._siteId = cursor.getInt(cursor.getColumnIndex(MantaEntry.MANTA_TABLE_COLUMN_SITE_ID));
        this._voyageId = cursor.getInt(cursor.getColumnIndex(MantaEntry.MANTA_TABLE_COLUMN_VOYAGE_ID));


    }


    //
    // GETTERS
    //


    // We provide individual public getter functions so that pieces of data can be read from
    // each Manta object.


    // Getting id
    public int getMantaId(){
        return this._mantaId;
    }


    // Getting date
    public String getMantaDate(){
        return this._mantaDate;
    }


    // At the moment, all dates are stored as YYYY-MM-DD strings, based on the structure available
    // to us in the GBRMPA Eye-On-The-Reef database exports. In future, this may be refined to
    // store date as UNIX time instead, to facilitate better range searches. For now, because we
    // often need to access the typeDate as a Date object for comparison or calculation, we provide
    // a method that: 1) checks that the typeDate is not null, and if it's OK, returns the typeDate
    // as a Date object.
    public Date getMantaDateAsDate() {


        SimpleDateFormat sdfDiveDate = new SimpleDateFormat( "yyyy-MM-dd");
        Date returnDate = null;


        try {


            returnDate = sdfDiveDate.parse( this._mantaDate );


        } catch (ParseException e) {
```

```
            e.printStackTrace();

        }

        return returnDate;

    }

    // Getting start latitude
    public double getMantaStartLat(){
        return this._mantaStartLat;
    }

    // Getting start longitude
    public double getMantaStartLong(){
        return this._mantaStartLong;
    }

    // Getting stop latitude
    public double getMantaStopLat(){
        return this._mantaStopLat;
    }

    // Getting stop longitude
    public double getMantaStopLong(){
        return this._mantaStopLong;
    }

    // Getting stop latitude
    public double getMantaMeanLat(){
        return this._mantaMeanLat;
    }

    // Getting stop longitude
    public double getMantaMeanLong(){
        return this._mantaMeanLong;
    }

    public LatLng getMantaMeanLatLng() { return new LatLng( this._mantaMeanLat, this._mantaMeanLong ); }

    // Getting cots
    public int getMantaCOTS(){
        return this._mantaCots;
    }

    // Getting scars
    public String getMantaScars(){
        return this._mantaScars;
    }
```

```
    // Getting site id
    public int getSiteId(){
        return this._siteId;
    }

    // Getting voyage id
    public int getVoyageId(){
        return this._voyageId;
    }



    //
    // DERIVED VALUES
    //

    // We also provide functions for derived values, like the total number of CoTS removed
    // and the catch per unit effort



    public long numberOfDaysSinceManta(){

        Date mantaDate = this.getMantaDateAsDate();

        Calendar todaysDate = Calendar.getInstance();
        long timeSinceManta = todaysDate.getTime().getTime() - mantaDate.getTime();
        long numberOfDaysSinceManta = TimeUnit.MILLISECONDS.toDays( timeSinceManta );

        return numberOfDaysSinceManta;

    }

    public boolean belowEcologicalThreshold(){

        boolean mantaCOTSAboveEcologicalThreshold = this._mantaCots <=
MainActivity.ecologicalThresholdMantaCOTS;
        boolean mantaScarsAboveEcologicalThreshold = this._mantaScars.equals( "a" );

        return ( mantaCOTSAboveEcologicalThreshold && mantaScarsAboveEcologicalThreshold  );

    }



    //
    // COMPARATORS
    //

    @Override
    public int compareTo(Manta manta) {

        Date thisMantaDate = null;
        Date mantaMantaDate = null;
```

```
    SimpleDateFormat sdfDiveDate = new SimpleDateFormat( "yyyy-MM-dd");

    try {

        thisMantaDate = sdfDiveDate.parse(this._mantaDate);
        mantaMantaDate = sdfDiveDate.parse(manta.getMantaDate());

    } catch (Exception e) {

        e.printStackTrace();

    }


    if ( mantaMantaDate.after( thisMantaDate ) )

        return -1;

    else if ( mantaMantaDate.before( thisMantaDate ) )

        return 1;

    else

        return 0;

}


public static Comparator<Manta> getMantaDateComparator() {

    return new Comparator<Manta>() {

        public int compare(Manta manta1, Manta manta2) {

            Date manta1Date = null;
            Date manta2Date = null;
            SimpleDateFormat sdfDiveDate = new SimpleDateFormat("yyyy-MM-dd");

            try {

                manta1Date = sdfDiveDate.parse(manta1.getMantaDate());
                manta2Date = sdfDiveDate.parse(manta2.getMantaDate());

            } catch (Exception e) {

                e.printStackTrace();

            }
```

```
                if ( manta1Date.after( manta2Date ) )
                    return -1;
                else if ( manta1Date.before( manta2Date ) )
                    return 1;
                else
                    return 0;
            }
        };
    }
    public static Comparator<Manta> getMantaCOTSNumberComparator() {
        return new Comparator<Manta>() {
            public int compare(Manta manta1, Manta manta2) {
                if ( manta1.getMantaCOTS() > manta2.getMantaCOTS() )
                    return -1;
                else if ( manta1.getMantaCOTS() < manta2.getMantaCOTS() )
                    return 1;
                else
                    return 0;
            }
        };
    }
}
```

## B.13 MantaList.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists;
```

```
import android.util.Log;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Manta;

//
// The purpose of these custom TypeList types, encompassing each of our custom Types, is to provide
// functionality related to the physical characteristics of the custom Types. For example, it
// makes sense to be able to call a method that returns the most recent Dives performed at each
// Site in a list of Mantas.
//
// There is an additional major benefit, in that we can store a HashMap of the indices of the
// TypeList when it is created. This allows us to write functions that can quickly find at item
// either by its index or by its Type identifier number. In the original data tables these two
// numbers are the same: i.e. the 131st Site has a SiteId of 131. However, in a generic SiteList
// this is not the case - if we make a list of Sites at a Reef, the Site with SiteId 131 might be
// the 10th item in that particular list.
//
// Implementation note: It is not clear to me whether it is more efficient to create a HashMap of
// < typeId, _typeListIndex >, as I do in this code, and then use that to look up the position of an
// item with a given typeId in the typeList, or whether I should make a HashMap <typeId, Type Object>
// directly. This way works for now, but we could do some performance testing later.
//
// Given the commonality between these TypeList classes, I should make a BaseTypeList class and
// then only include the code specific to each Type within the TypeClass, but I haven't got
// around to this simplification yet.
//

public class MantaList implements Iterable<Manta> {

    private List<Manta> _mantaList = new ArrayList<Manta>();
    private HashMap<Integer, Integer> _mantaListIndex = new HashMap< Integer, Integer >();

    //
    // CONSTRUCTORS
    //

    // Empty constructor
    public MantaList(){
    }
```

```
// Constructor
public MantaList( List<Manta> mantaList ){

    this._mantaList = mantaList;

    for ( int i = 0; i < mantaList.size(); i++ ) {

        this._mantaListIndex.put( mantaList.get(i).getMantaId(), i );

    }

}

//
// SETTERS
//

// This function simply adds a manta to MantaList.
public void add( Manta manta ) {

    this._mantaList.add( manta );

    this._mantaListIndex.put( manta.getMantaId(), this._mantaList.size() - 1 );

}

public void clear() {

    this._mantaList.clear();

    this._mantaListIndex.clear();

}

//
// GETTERS
//

public Manta get( int i ) {

    return this._mantaList.get( i );

}

// Get entire Manta List
public List<Manta> getMantaListCopy(){

    // Only ever return a copy of the _mantaList so we don't muck up the ordering of the
    // <key,value> pairs in _mantaListIndex
```

```
    List<Manta> mantaListCopy = new ArrayList( this._mantaList );

    return mantaListCopy;

}

// Get Manta from mantaList by mantaId
public Manta getMantaByMantaId( int mantaId ){

    int mantaTableIndex  = this._mantaListIndex.get( mantaId );

    return _mantaList.get( mantaTableIndex );

}

// Get a list of Mantas from mantaList by a list of mantaIds
public MantaList getMantaByMantaId( List<Integer> mantaIds ){

    MantaList returnMantas = new MantaList();

    for ( int mantaId : mantaIds ) {

        int mantaIndex = this._mantaListIndex.get( mantaId );

        returnMantas.add( this._mantaList.get( mantaIndex ) );

    }

    return returnMantas;

}

public MantaList getMantasOnMostRecentVoyage(){

    if ( this._mantaList.isEmpty() ) {

        return ( new MantaList ( new ArrayList<Manta>() ) );

    } else {

        // Get a copy of the underlying list of Dives, so we don't sort the actual DiveList
        List<Manta> mantaListCopy = new ArrayList<>(this._mantaList);

        // Sort the copy
        Collections.sort( mantaListCopy, Manta.getMantaDateComparator() );

        // Find the voyageId of the most recent Dive
        int voyageId = mantaListCopy.get(0).getVoyageId();

        // Return a list of the Dive with that voyageId
```

```
            return this.getMantasByVoyageId( voyageId );

        }

    }

    public MantaList getMantasByVoyageId( int voyageId ){

        MantaList returnMantaList = new MantaList();

        for ( Manta manta : this._mantaList ){

            if ( manta.getVoyageId() == voyageId ){

                returnMantaList.add( manta );

            }

        }

        return returnMantaList;

    }

    public MantaList getMantasBySiteId( int siteId ){

        MantaList returnMantaList = new MantaList();

        for ( Manta manta : this._mantaList ){

            if ( manta.getSiteId() == siteId ){

                returnMantaList.add( manta );

            }

        }

        return returnMantaList;

    }

    //
    // DERIVED VALUES
    //

    // We also provide functions for derived values, like the total number of CoTS removed
    // and the catch per unit effort achieved across a list of Type

    public boolean isEmpty() {
```

```
    return this._mantaList.isEmpty();

}

public int size() {

    return this._mantaList.size();

}


public Integer getTotalCOTS(){

    if ( this._mantaList.isEmpty() ) {

        return 0;

    } else {

        int totalCOTS = 0;

        for ( Manta manta : this._mantaList ) {

            totalCOTS += manta.getMantaCOTS();

        }

        return totalCOTS;

    }

}

public String getTotalScars() {

    if ( this._mantaList.isEmpty() ) {

        return "a";

    } else {

        String totalScars = "a";

        for ( Manta manta : this._mantaList ) {

            if ( totalScars.equals( "a" ) && manta.getMantaScars().equals( "a" ) ){

                totalScars = "a";
```

```
                } else if ( totalScars.equals( "a" ) && manta.getMantaScars().equals( "p" ) ) {
                    totalScars = "p";
                } else if ( totalScars.equals( "p" ) && manta.getMantaScars().equals( "a" ) ) {
                    totalScars = "p";
                } else {
                    totalScars = "c";
                }
            }
            return totalScars;
        }
    }
    @Nullable
    public Date getMostRecentMantaDate() {
        if ( this._mantaList.isEmpty() ) {
            return null;
        } else {
            return getMostRecentManta().getMantaDateAsDate();
        }
    }
    @Nullable
    public Integer getMostRecentMantaVoyageId() {
        if ( this._mantaList.isEmpty() ) {
            return null;
//            return ( new DiveList ( new ArrayList<Dive>() ) );
        } else {
            return getMostRecentManta().getVoyageId();
```

```
        }

    }

    @Nullable
    public Manta getMostRecentManta() {

        if ( this._mantaList.isEmpty() ) {

            return null;

//            return ( new DiveList ( new ArrayList<Dive>() ) );

        } else {

            // Get a copy of the underlying list of Dives, so we don't sort the actual DiveList
            List<Manta> mantaListCopy = new ArrayList<>( this._mantaList );

            // Sort the copy
            Collections.sort( mantaListCopy, Manta.getMantaDateComparator() );

            // Find the voyageId of the most recent Dive
            int voyageId = mantaListCopy.get( 0 ).getVoyageId();

            // Return a list of the Dive with that voyageId
            return this.getMantasByVoyageId( voyageId ).get( 0 );

        }

    }


    // Get Manta from mantaList by mantaId
    public int getMantaListIndexByMantaId( int mantaId ){

        return this._mantaListIndex.get( mantaId );

    }

    public Manta getMantaByIndex( int index ){

        return _mantaList.get( index );

    }

    //
    // Iterators
    //

    @NonNull
```

```
    @Override
    public Iterator<Manta> iterator() {

        return this._mantaList.iterator();

    }

    // This function should only be used on a MantaList describing Mantas conducted at the same
    // Site on the same Voyage. At the moment we just post a Log message - in future we should
    // Throw and exception.
    public Manta createCompositeManta (){

        int mantaId;
        String mantaDate;
        double mantaStartLat = 0;
        double mantaStartLong = 0;
        double mantaStopLat = 0;
        double mantaStopLong = 0;
        double mantaMeanLat = 0;
        double mantaMeanLong = 0;
        int mantaCots = 0;
        String mantaScars = "a";
        int siteId;
        int voyageId;

        if ( !this._mantaList.isEmpty() ) {

            mantaDate = this._mantaList.get(0).getMantaDate();
            siteId = this._mantaList.get(0).getSiteId();
            voyageId = this._mantaList.get(0).getVoyageId();

            mantaCots = this.getTotalCOTS();
            mantaScars = this.getTotalScars();

            // This assignment of lats and longs depends on the order in which mantas are listed and
            // the relative lat and long of the points - but as we don't use this for anything at the
            // moment, it's not important that it's deterministic.
            mantaStartLat = this._mantaList.get(0).getMantaStartLat();
            mantaStartLong = this._mantaList.get(0).getMantaStartLong();
            mantaStopLat = this._mantaList.get(this._mantaList.size() - 1).getMantaStartLat();
            mantaStopLong = this._mantaList.get(this._mantaList.size() - 1).getMantaStartLong();
            mantaMeanLat = (mantaStartLat + mantaStopLat) / 2;
            mantaMeanLong = (mantaStartLong + mantaStopLong) / 2;


            for (Manta manta : this._mantaList) {

                if ((manta.getSiteId() != siteId) || (manta.getVoyageId() != voyageId)) {

                    Log.d("CCC_MANTA", "Mantas should not have been combined because they span multiple
Sites or Voyages");
```

```
                }

            }

            return new Manta(-1, mantaDate, mantaStartLat, mantaStartLong, mantaStopLat, mantaStopLong,
mantaMeanLat, mantaMeanLong, mantaCots, mantaScars, siteId, voyageId);

        } else {

            return new Manta();

        }

    }


}
```

## B.14 Site.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types;

import android.content.Context;
import android.database.Cursor;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Locale;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.R;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.SiteEntry;

/**
 * Created by fle125 on 31/03/2017.
 */

public class Site {

    // Private Variables
    // These are the components intrinsic to a Site, rather than values that should be derived
    // from the Dive or Voyage that reports its activities relative to a Site.
    // For instance - the Site latitude and longitude are intrinsic to a Site.
    // On the other hand, the number of CoTS removed on a specific Dive at that Site are a
    // Dive characteristic. We do not want to record the number of CoTS removed at a given Site,
    // but we may want to provide a method that can return the total number of CoTS ever removed
    // at that Site.
```

```
    private int _siteId;
    private String _siteName;
    private double _siteLatitude;
    private double _siteLongitude;
    private int _reefId;

    //
    // CONSTRUCTORS
    //

    // Empty constructor
    public Site(){
    }

    // Constructor
    public Site(int siteId, String siteName, double siteLatitude, double siteLongitude, int reefId ){

        this._siteId = siteId;
        this._siteName = siteName;
        this._siteLatitude = siteLatitude;
        this._siteLongitude = siteLongitude;
        this._reefId = reefId;

    }

    // Constructor
    public Site(Cursor cursor) {

        this._siteId = cursor.getInt(cursor.getColumnIndex(SiteEntry.SITE_TABLE_COLUMN_ID));
        this._siteName = cursor.getString(cursor.getColumnIndex(SiteEntry.SITE_TABLE_COLUMN_SITE_NAME));
        this._siteLatitude = cursor.getDouble(cursor.getColumnIndex(SiteEntry.SITE_TABLE_COLUMN_LATITUDE));
        this._siteLongitude =
cursor.getDouble(cursor.getColumnIndex(SiteEntry.SITE_TABLE_COLUMN_LONGITUDE));
        this._reefId = cursor.getInt(cursor.getColumnIndex(SiteEntry.SITE_TABLE_COLUMN_REEF_ID));

    }

    //
    // GETTERS
    //

    // We provide individual public getter functions so that pieces of data can be read from
    // each Site object.

    // Getting id
    public int getSiteId(){
        return this._siteId;
    }

    // Getting siteName
    public String getSiteName(){
```

```
        return this._siteName;
    }

    // Getting latitude
    public double getSiteLatitude(){
        return this._siteLatitude;
    }

    // Getting longitude
    public double getSiteLongitude(){
        return this._siteLongitude;
    }

    // Getting Reef Id
    public Integer getReefId(){
        return this._reefId;
    }

}
```

## B.15 SiteList.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists;

import androidx.annotation.NonNull;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.Site;

//
// The purpose of these custom TypeList types, encompassing each of our custom Types, is to provide
// functionality related to the physical characteristics of the custom Types. For example, it
// makes sense to be able to call a method that returns the most recent Dives performed at each
// Site in a list of Dives.
//
// There is an additional major benefit, in that we can store a HashMap of the indices of the
// TypeList when it is created. This allows us to write functions that can quickly find at item
// either by its index or by its Type identifier number. In the original data tables these two
// numbers are the same: i.e. the 131st Site has a SiteId of 131. However, in a generic SiteList
// this is not the case - if we make a list of Sites at a Reef, the Site with SiteId 131 might be
// the 10th item in that particular list.
//
// Implementation note: It is not clear to me whether it is more efficient to create a HashMap of
// < typeId, _typeListIndex >, as I do in this code, and then use that to look up the position of an
// item with a given typeId in the typeList, or whether I should make a HashMap <typeId, Type Object>
// directly. This way works for now, but we could do some performance testing later.
```

```
//
// Given the commonality between these TypeList classes, I should make a BaseTypeList class and
// then only include the code specific to each Type within the TypeClass, but I haven't got
// around to this simplification yet.
//
// Note that we do not extend one of Android's list classes because: 1) that is not considered
// best practice, and 2) it would require a lot of boilerplate overridden function definitions.
//

public class SiteList implements Iterable<Site> {

    private List<Site> _siteList = new ArrayList<Site>();
    private HashMap<Integer, Integer> _siteListIndex = new HashMap< Integer, Integer >();

    private boolean bufferDiveChanges = true;
    private boolean bufferMantaChanges = true;
    private HashMap<Integer, DiveList> bufferedMostRecentDiveAtEachSite = new HashMap< Integer, DiveList
>();
    private HashMap<Integer, List<Integer>> bufferedMostRecentMantasAtEachSite = new HashMap< Integer,
List<Integer> >();

    //
    // CONSTRUCTORS
    //

    // Empty constructor
    public SiteList(){

        this.bufferDiveChanges = true;
        this.bufferMantaChanges = true;

    }

    // Constructor
    public SiteList( List<Site> siteList ){

        this._siteList = siteList;

        for ( int i = 0; i < siteList.size(); i++ ) {

            this._siteListIndex.put( siteList.get(i).getSiteId(), i );

        }

        this.bufferDiveChanges = true;
        this.bufferMantaChanges = true;

    }

    //
    // SETTERS
```

```
//

public void add( Site site ) {

    this._siteList.add( site );

    this._siteListIndex.put( site.getSiteId(), this._siteList.size() - 1 );

    this.bufferDiveChanges = true;
    this.bufferMantaChanges = true;

}

public void clear() {

    this._siteList.clear();

    this._siteListIndex.clear();

    this.bufferDiveChanges = true;
    this.bufferMantaChanges = true;

}

//
// GETTERS
//

public void getSiteBySiteId(){

}

// Get Site from siteList by siteId
public Site getSiteBySiteId( int siteId ){

    int siteTableIndex  = this._siteListIndex.get( siteId );

    return _siteList.get( siteTableIndex );

}

// Get a list of Sites from siteList by a list of siteIds
public SiteList getSiteBySiteId( List<Integer> siteIds ){

    SiteList returnSites = new SiteList();

    for ( int siteId : siteIds ) {

        int siteIndex = this._siteListIndex.get( siteId );
```

```
            returnSites.add( this._siteList.get( siteIndex ) );

        }

        return returnSites;

    }

    // Get Site from siteList by siteId
    public int getSiteListIndexBySiteId( int siteId ){

        return this._siteListIndex.get( siteId );

    }

    public void updateDiveAndMantaChanges(){

        this.bufferDiveChanges = true;
        this.bufferMantaChanges = true;

    }

    //
    // DERIVED VALUES
    //

    // We also provide functions for derived values, like the total number of CoTS removed
    // and the catch per unit effort achieved across a list of Type

    public Integer size() {

        return this._siteList.size();

    }

    public boolean isEmpty(){

        return this._siteList.isEmpty();

    }

    //
    // Iterators
    //

    @NonNull
    @Override
    public Iterator<Site> iterator() {

        return this._siteList.iterator();
```

```
    }

    public int findSiteIdBySiteName( String siteName ){

        int returnId = 0;

        for ( Site site : _siteList ){

            if ( site.getSiteName().equals( siteName ) ){

                returnId = site.getSiteId();

            }

        }

        return returnId;

    }

}
```

## B.16 SitePolygon.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types;

import com.google.android.gms.maps.model.LatLng;

import java.util.ArrayList;
import java.util.List;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists.SitePolygonPointList;

/**
 * Created by fle125 on 31/03/2017.
 */

public class SitePolygon {

    // Private Variables
    // These are the components intrinsic to a SitePolygon, rather than values that should be
    // derived from the associated Site.

    private List<Integer> _sitePolygonPointOrderList;
    private List<LatLng> _sitePolygonPointLatLngs;
    private Integer _siteId;

    //
    // CONSTRUCTORS
```

```
    //


    // Empty constructor
    public SitePolygon(){
    };


    // Constructor based on passing all required values
    public SitePolygon(int siteId, List<Integer> sitePolygonPointOrderList, List<Double>
sitePolygonPointLatitudeList, List<Double> sitePolygonPointLongitudeList ){


        this._siteId = siteId;


        List<LatLng> latLngList = null;


        for (int i = 0;i<sitePolygonPointOrderList.size();i++) {
            latLngList.add( new LatLng( sitePolygonPointLatitudeList.get( i ),
sitePolygonPointLatitudeList.get( i ) ) );
        }
        this._sitePolygonPointLatLngs = latLngList;


    }


    public SitePolygon(List<SitePolygonPoint> sitePolygonPointList){


        this._siteId = sitePolygonPointList.get(0).getSiteId();


        List<LatLng> latLngList = new ArrayList<LatLng>();


        for (SitePolygonPoint sitePolygonPoint: sitePolygonPointList) {
            latLngList.add( new LatLng( sitePolygonPoint.getSitePolygonPointLatitude(),
sitePolygonPoint.getSitePolygonPointLongitude() ) );
        }


        this._sitePolygonPointLatLngs = latLngList;


    }


    public SitePolygon( SitePolygonPointList sitePolygonPointList ){


        this._siteId = sitePolygonPointList.get(0).getSiteId();


        List<LatLng> latLngList = new ArrayList<LatLng>();


        for (SitePolygonPoint sitePolygonPoint: sitePolygonPointList) {
            latLngList.add( new LatLng( sitePolygonPoint.getSitePolygonPointLatitude(),
sitePolygonPoint.getSitePolygonPointLongitude() ) );
        }


        this._sitePolygonPointLatLngs = latLngList;


    }
```

```
    //
    // GETTERS
    //

    // We provide individual public getter functions so that pieces of data can be read from
    // each Dive object.

    // Getting id
    public int getSiteId(){
        return this._siteId;
    }

    // Getting site polygon
    public List<LatLng> getSitePolygonPoints(){
        return this._sitePolygonPointLatLngs;
    }

}
```

## B.17 SitePolygonList.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists;

import androidx.annotation.NonNull;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.SitePolygon;

//
// The purpose of these custom TypeList types, encompassing each of our custom Types, is to provide
// functionality related to the physical characteristics of the custom Types. For example, it
// makes sense to be able to call a method that returns the most recent Dives performed at each
// Site in a list of Dives.
//
// There is an additional major benefit, in that we can store a HashMap of the indices of the
// TypeList when it is created. This allows us to write functions that can quickly find at item
// either by its index or by its Type identifier number. In the original data tables these two
// numbers are the same: i.e. the 131st Site has a SiteId of 131. However, in a generic SiteList
// this is not the case - if we make a list of Sites at a Reef, the Site with SiteId 131 might be
// the 10th item in that particular list.
//
// Implementation note: It is not clear to me whether it is more efficient to create a HashMap of
// < typeId, _typeListIndex >, as I do in this code, and then use that to look up the position of an
// item with a given typeId in the typeList, or whether I should make a HashMap <typeId, Type Object>
// directly. This way works for now, but we could do some performance testing later.
//
```

```
// Given the commonality between these TypeList classes, I should make a BaseTypeList class and
// then only include the code specific to each Type within the TypeClass, but I haven't got
// around to this simplification yet.
//

public class SitePolygonList implements
        Iterable<SitePolygon> {

    private List<SitePolygon> _sitePolygonList = new ArrayList<>();
    private HashMap<Integer, Integer> _sitePolygonListIndex = new HashMap< Integer, Integer >();

    //
    // CONSTRUCTORS
    //

    // Empty constructor
    public SitePolygonList(){
    }

    // Constructor
    public SitePolygonList( List<SitePolygon> sitePolygonList ){

        this._sitePolygonList = sitePolygonList;

        for ( int i = 0; i < sitePolygonList.size(); i++ ) {

            this._sitePolygonListIndex.put( sitePolygonList.get( i ).getSiteId(), i );

        }

    }

    //
    // SETTERS
    //

    public void add( SitePolygon sitePolygon ) {

        this._sitePolygonList.add( sitePolygon );

        this._sitePolygonListIndex.put( sitePolygon.getSiteId(), this._sitePolygonList.size() - 1 );

    }

    public void clear() {

        this._sitePolygonList.clear();

        this._sitePolygonListIndex.clear();
```

```
}


//
// GETTERS
//

// Get SitePolygon from sitePolygonList by siteId
public SitePolygon getSitePolygonBySiteId( int siteId ){

    Integer sitePolygonTableIndex  = this._sitePolygonListIndex.get( siteId );

    if ( sitePolygonTableIndex != null ) {

        return _sitePolygonList.get(sitePolygonTableIndex);

    } else {

        return null;

    }

}

// Get a list of SitePolygon from sitePolygonList by a list of siteIds
public SitePolygonList getSitePolygonsBySiteIds( List<Integer> siteIds ){

    SitePolygonList returnSitePolygonList = new SitePolygonList();

    for ( int siteId : siteIds ) {

        int sitePolygonIndex = this._sitePolygonListIndex.get( siteId );

        returnSitePolygonList.add( this._sitePolygonList.get( sitePolygonIndex ) );

    }

    return returnSitePolygonList;

}

// Get copy of SitePolygonList
public List<SitePolygon> getSitePolygonListCopy(){

    // Only ever return a copy of the _mantaList so we don't muck up the ordering of the
    // <key,value> pairs in _mantaListIndex

    List<SitePolygon> sitePolygonListCopy = new ArrayList<SitePolygon>( this._sitePolygonList );

    return sitePolygonListCopy;
```

```
    }

    public SitePolygon get( int i ) {

        return this._sitePolygonList.get( i );

    }

    //
    // DERIVED VALUES
    //

    public boolean isEmpty() {

        return this._sitePolygonList.isEmpty();

    }

    //
    // Iterators
    //

    @NonNull
    @Override
    public Iterator<SitePolygon> iterator() {

        return this._sitePolygonList.iterator();

    }

}
```

## B.18 SitePolygonPoint.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types;

import android.database.Cursor;

import java.util.Comparator;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.SitePolygonsEntry;

/**
 * Created by fle125 on 31/03/2017.
 */

public class SitePolygonPoint implements Comparable<SitePolygonPoint> {

    // Private Variables
```

```
    // These are the components intrinsic to a SitePolygonPoint, rather than values that should be
    // derived from the Site itself.

    private int _sitePolygonsId;
    private int _sitePolygonPointOrder;
    private double _sitePolygonPointLatitude;
    private double _sitePolygonPointLongitude;
    private int _siteId;

    //
    // CONSTRUCTORS
    //

    // Empty constructor
    public SitePolygonPoint(){
    }

    // Constructor based on passing all required values
    public SitePolygonPoint(int sitePolygonsId, int sitePolygonPointOrder, double sitePolygonPointLatitude,
double sitePolygonPointLongitude, int siteId ){

        this._sitePolygonsId = sitePolygonsId;
        this._sitePolygonPointOrder = sitePolygonPointOrder;
        this._sitePolygonPointLatitude = sitePolygonPointLatitude;
        this._sitePolygonPointLongitude = sitePolygonPointLongitude;
        this._siteId = siteId;

    }

    // Constructor based on passing a single Cursor to a SiteEntry
    public SitePolygonPoint(Cursor cursor) {

        this._sitePolygonsId =
cursor.getInt(cursor.getColumnIndex(SitePolygonsEntry.SITE_POLYGONS_TABLE_COLUMN_ID));
        this._sitePolygonPointOrder =
cursor.getInt(cursor.getColumnIndex(SitePolygonsEntry.SITE_POLYGONS_TABLE_COLUMN_POINT_ORDER));
        this._sitePolygonPointLatitude =
cursor.getDouble(cursor.getColumnIndex(SitePolygonsEntry.SITE_POLYGONS_TABLE_COLUMN_POINT_LATITUDE));
        this._sitePolygonPointLongitude =
cursor.getDouble(cursor.getColumnIndex(SitePolygonsEntry.SITE_POLYGONS_TABLE_COLUMN_POINT_LONGITUDE));
        this._siteId =
cursor.getInt(cursor.getColumnIndex(SitePolygonsEntry.SITE_POLYGONS_TABLE_COLUMN_SITE_ID));

    }

    //
    // GETTERS
    //

    // We provide individual public getter functions so that pieces of data can be read from
    // each SitePolygonPoint object.

    // Getting id
```

```
    public int getSitePolygonsId(){
        return this._sitePolygonsId;
    }

    // Getting SiteName
    public int getSitePolygonPointOrder(){
        return this._sitePolygonPointOrder;
    }

    // Getting SiteId
    public double getSitePolygonPointLatitude(){
        return this._sitePolygonPointLatitude;
    }

    // Getting latitude
    public double getSitePolygonPointLongitude(){ return this._sitePolygonPointLongitude; }

    // Getting longitude
    public int getSiteId(){ return this._siteId; }



    //
    // COMPARATORS
    //

    @Override
    public int compareTo( SitePolygonPoint sitePolygonPoint ) {

        if ( sitePolygonPoint.getSitePolygonPointOrder() < this.getSitePolygonPointOrder() )

            return -1;

        else if ( sitePolygonPoint.getSitePolygonPointOrder() < this.getSitePolygonPointOrder()  )

            return 1;

        else

            return 0;

    }

    public static Comparator<SitePolygonPoint> getPolygonPointComparator() {

        return new Comparator<SitePolygonPoint>() {

            public int compare(SitePolygonPoint sitePolygonPoint1, SitePolygonPoint sitePolygonPoint2) {

                if ( sitePolygonPoint1.getSitePolygonPointOrder() <
sitePolygonPoint2.getSitePolygonPointOrder() )
```

```
                    return -1;

                else if ( sitePolygonPoint1.getSitePolygonPointOrder() >
sitePolygonPoint2.getSitePolygonPointOrder()  )

                    return 1;

                else

                    return 0;


            }

        };

    }

}
```

## B.19 SitePolygonPointList.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.typeLists;

import androidx.annotation.NonNull;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.SitePolygon;
import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types.SitePolygonPoint;

//
// The purpose of these custom TypeList types, encompassing each of our custom Types, is to provide
// functionality related to the physical characteristics of the custom Types. For example, it
// makes sense to be able to call a method that returns the most recent Dives performed at each
// Site in a list of Dives.
//
// There is an additional major benefit, in that we can store a HashMap of the indices of the
// TypeList when it is created. This allows us to write functions that can quickly find at item
// either by its index or by its Type identifier number. In the original data tables these two
// numbers are the same: i.e. the 131st Site has a SiteId of 131. However, in a generic SiteList
// this is not the case - if we make a list of Sites at a Reef, the Site with SiteId 131 might be
// the 10th item in that particular list.
```

```
//
// Implementation note: It is not clear to me whether it is more efficient to create a HashMap of
// < typeId, _typeListIndex >, as I do in this code, and then use that to look up the position of an
// item with a given typeId in the typeList, or whether I should make a HashMap <typeId, Type Object>
// directly. This way works for now, but we could do some performance testing later.
//
// Given the commonality between these TypeList classes, I should make a BaseTypeList class and
// then only include the code specific to each Type within the TypeClass, but I haven't got
// around to this simplification yet.
//

// This may seem a little pointless, but the goal here is to provide a couple of simple classes
// that allow all the SitePolygonPoints belonging to a specific siteId to be extracted, and
// also to extract them in sorted order

public class SitePolygonPointList implements
        Iterable<SitePolygonPoint> {

    private List<SitePolygonPoint> _sitePolygonPointList = new ArrayList<>();

    //
    // CONSTRUCTORS
    //

    // Empty constructor
    public SitePolygonPointList(){
    }

    // Constructor
    public SitePolygonPointList(List<SitePolygonPoint> sitePolygonPointList ){

        this._sitePolygonPointList = sitePolygonPointList;

    }

    // Constructor
    public SitePolygonPointList( SitePolygonPoint sitePolygonPoint ){

        List<SitePolygonPoint> sitePolygonPointList = new ArrayList<>();
        sitePolygonPointList.add( sitePolygonPoint );

        this._sitePolygonPointList = sitePolygonPointList;

    }

    //
    // SETTERS
    //

    public void add( SitePolygonPoint sitePolygonPoint ) {
```

```
        this._sitePolygonPointList.add( sitePolygonPoint );

    }

    public void addAll( SitePolygonPointList sitePolygonPointList ) {

        this._sitePolygonPointList.addAll( sitePolygonPointList.getSitePolygonPointsList() );

    }

    public void clear() {

        this._sitePolygonPointList.clear();

    }

    //
    // GETTERS
    //

    // Get copy of SitePolygonList
    public SitePolygonPointList getSitePolygonPointListCopy(){

        // Only ever return a copy of the _mantaList so we don't muck up the ordering of the
        // <key,value> pairs in _mantaListIndex

        SitePolygonPointList sitePolygonPointListCopy = new SitePolygonPointList(
this._sitePolygonPointList);

        return sitePolygonPointListCopy;

    }

    public SitePolygonPoint get( int i ) {

        return this._sitePolygonPointList.get( i );

    }

    public SitePolygonPointList getSitePolygonPointsBySiteId( int siteId ){

        SitePolygonPointList returnSitePolygonPointList = new SitePolygonPointList();

        for ( SitePolygonPoint sitePolygonPoint : this._sitePolygonPointList ){

            if ( sitePolygonPoint.getSiteId() == siteId ){

                returnSitePolygonPointList.add( sitePolygonPoint );

            }
```

```
        }

        return returnSitePolygonPointList;

    }

    public List<SitePolygonPoint> getSitePolygonPointsList(){

        return this._sitePolygonPointList;

    }

    //
    // DERIVED VALUES
    //

    // We also provide functions for derived values, like the total number of CoTS removed
    // and the catch per unit effort achieved across a list of Type

    public boolean isEmpty() {

        return this._sitePolygonPointList.isEmpty();

    }

    // Returns a sorted copy of the list - we could perhaps avoid the copy in future.
    public SitePolygonPointList getSitePolygonPointsBySiteIdSorted( int siteId ){

        SitePolygonPointList sitePolygonPointsBySiteId = this.getSitePolygonPointsBySiteId( siteId );

        List<SitePolygonPoint> sitePolygonPointsBySiteIdList =
sitePolygonPointsBySiteId.getSitePolygonPointsList();

        Collections.sort( sitePolygonPointsBySiteIdList, SitePolygonPoint.getPolygonPointComparator() );

        return ( new SitePolygonPointList( sitePolygonPointsBySiteIdList ) );

    }

    public List<Integer> getSiteIdsOfSitePolygonPointsInList(){

        List<Integer> returnSiteIdList = new ArrayList<>();

        Set<Integer> siteIdsNoRepeats = new HashSet<>();

        for ( SitePolygonPoint sitePolygonPoint : this._sitePolygonPointList ){

            siteIdsNoRepeats.add( sitePolygonPoint.getSiteId() );

        }
```

```
        returnSiteIdList.addAll( siteIdsNoRepeats );

        return returnSiteIdList;

    }

    public SitePolygonList getAllSitePolygons(){

        HashMap<Integer, SitePolygonPointList> sitePolygonPointLists = new HashMap<>();

        SitePolygonList returnSitePolygonList = new SitePolygonList();

        // Based on the premise that most of hte PolygonPoint should be in order of the Polgyons
        // they belong to, and that updating the HashMap is expensive, we cycle through all the
        // sitePolygonPoints in the list, adding runs of sitePolygonPoints with the same siteId
        // to a temporary list, and we only update the HashMap when the next siteId is different
        // from the preceding one.
        int lastSiteId = this._sitePolygonPointList.get(0).getSiteId();
        SitePolygonPointList sitePolygonPointList = new SitePolygonPointList();

        for ( SitePolygonPoint sitePolygonPoint : this ){

            int siteId = sitePolygonPoint.getSiteId();

            if ( siteId == lastSiteId ){

                sitePolygonPointList.add( sitePolygonPoint );

            } else {

                if ( sitePolygonPointLists.containsKey( lastSiteId ) ){

                    sitePolygonPointLists.get( lastSiteId ).addAll( sitePolygonPointList );

                } else {
//
                    sitePolygonPointLists.put( lastSiteId, sitePolygonPointList );

                }

                lastSiteId = siteId;

                sitePolygonPointList = new SitePolygonPointList();

                sitePolygonPointList.add( sitePolygonPoint );

            }
```

```
        }

        // Make sure you add the last Sites details
        if ( sitePolygonPointLists.containsKey( lastSiteId ) ){

            sitePolygonPointLists.get( lastSiteId ).addAll( sitePolygonPointList );

        } else {
//
            sitePolygonPointLists.put( lastSiteId, sitePolygonPointList );

        }



        // Now, turn each of those into a SitePolygon
        for ( Map.Entry<Integer,SitePolygonPointList> sitePolygonPointListEntry :
sitePolygonPointLists.entrySet() ){

            SitePolygonPointList sitePolygonPointList2 = sitePolygonPointListEntry.getValue();

            List<SitePolygonPoint> sitePolygonPointListList =
sitePolygonPointList2.getSitePolygonPointsList();

            Collections.sort( sitePolygonPointListList, SitePolygonPoint.getPolygonPointComparator() );

            returnSitePolygonList.add( new SitePolygon( sitePolygonPointListList ) );

        }

        return returnSitePolygonList;

    }

    //
    // Iterators
    //

    @NonNull
    @Override
    public Iterator<SitePolygonPoint> iterator() {

        return this._sitePolygonPointList.iterator();

    }

}
```

## B.20 Reef.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types;

import android.database.Cursor;

import java.util.Locale;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.ReefEntry;

/**
 * Created by fle125 on 31/03/2017.
 */

public class Reef {

    // Private Variables
    // These are the components intrinsic to a Reef, rather than values that should be derived
    // from the Dive or Voyage that reports its activities relative to a Reef.
    // For instance - the Reef latitude and longitude are intrinsic to a Reef.
    // On the other hand, the number of CoTS removed on a specific Dive at that Reef are a
    // Dive characteristic. We do not want to record the number of CoTS removed at a given Reef,
    // but we may want to provide a method that can return the total number of CoTS ever removed
    // at that Reef.

    private int _reefId;
    private String _reefReefName;
    private String _reefReefId;
    private double _reefLatitude;
    private double _reefLongitude;

    // Empty constructor
    public Reef(){
    }

    // Constructor based on passing all required values
    public Reef(int reefId, String reefReefName, String reefReefId, double reefLatitude, double
reefLongitude ){

        this._reefId = reefId;
        this._reefReefName = reefReefName;
        this._reefReefId = reefReefId;
        this._reefLatitude = reefLatitude;
        this._reefLongitude = reefLongitude;

    }

    // Constructor based on passing a single Cursor to a SiteEntry
    public Reef(Cursor cursor) {
```

```
        this._reefId = cursor.getInt(cursor.getColumnIndex(ReefEntry.REEF_TABLE_COLUMN_ID));
        this._reefReefName =
cursor.getString(cursor.getColumnIndex(ReefEntry.REEF_TABLE_COLUMN_REEF_NAME));
        this._reefReefId = cursor.getString(cursor.getColumnIndex(ReefEntry.REEF_TABLE_COLUMN_REEF_ID));
        this._reefLatitude = cursor.getDouble(cursor.getColumnIndex(ReefEntry.REEF_TABLE_COLUMN_LATITUDE));
        this._reefLongitude =
cursor.getDouble(cursor.getColumnIndex(ReefEntry.REEF_TABLE_COLUMN_LONGITUDE));

    }

    // We do provide individual public getter functions so that pieces of data can be read from
    // each Rhis object.

    // Getting id
    public int getReefId(){
        return this._reefId;
    }

    // Getting reefName
    public String getReefName(){
        return this._reefReefName;
    }

    // Getting reefId
    public String getReefReefId(){
        return this._reefReefId;
    }

    // Getting latitude
    public double getReefLatitude(){ return this._reefLatitude; }

    // Getting longitude
    public double getReefLongitude(){ return this._reefLongitude; }

    public String formattedIcon() {
        return ( "R" );
    }

    public String formattedHeading() {
        return ( this._reefReefName );
    }

    public String formattedSubHeading() {
        return ( "Lat: " + String.format(Locale.US, "%.2f", this.getReefLatitude()) + ", Long: " +
String.format(Locale.US, "%.2f", this.getReefLongitude()) );
    }

//    public int totalRemoved() {
//
//        List<Site> SiteList = getSites();
//
```

```
//        int totalRemovedResult = 0;
//
//        for ( Site site : SiteList ) {
//
//            List<Dive> containingDiveList = site.getContainingDives();
//
//            for ( Dive dive : containingDiveList ) {
//                totalRemovedResult += dive.totalCoTS();
//            }
//
//        }
//
//        return totalRemovedResult;
//
//    }

//    public int avoidedDamage() {
//
//        List<Site> SiteList = getSites();
//
//        int avoidedDamageResult = 0;
//
//        for ( Site site : SiteList ) {
//
//            List<Dive> containingDiveList = site.getContainingDives();
//
//            for ( Dive dive : containingDiveList ) {
//                avoidedDamageResult += dive.avoidedDamage();
//            }
//
//        }
//
//        return avoidedDamageResult;
//
//    }


//    public String formattedTotalRemoved() {
//
//        return FormatNumbers.formatNumber( totalRemoved() );
//
//    }
//
//    public String formattedAvoidedDamage() {
//
//        return FormatNumbers.formatNumber( avoidedDamage() );
//
//    }

}
```

## B.21 ReefPolygon.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types;


import com.google.android.gms.maps.model.LatLng;


import java.util.ArrayList;
import java.util.List;


/**
 * Created by fle125 on 31/03/2017.
 */


public class ReefPolygon {


    // Private Variables
    // These are the components intrinsic to a ReefPolygon, rather than values that should be
    // derived from the Reef.


    private List<Integer> _reefPolygonPointOrderList;
    private List<LatLng> _reefPolygonPointLatLngs;
    private Integer _reefId;


    //
    // CONSTRUCTORS
    //


    // Empty constructor
    public ReefPolygon(){
    };


    // Constructor based on passing all required values
    public ReefPolygon(int reefId, List<Integer> reefPolygonPointOrderList, List<Double>
reefPolygonPointLatitudeList, List<Double> reefPolygonPointLongitudeList ){


        this._reefId = reefId;


        List<LatLng> latLngList = new ArrayList<>();


        for (int i = 0;i<reefPolygonPointOrderList.size();i++) {
            latLngList.add( new LatLng( reefPolygonPointLatitudeList.get( i ),
reefPolygonPointLongitudeList.get( i ) ) );
        }
        this._reefPolygonPointLatLngs = latLngList;


    }


    // Constructor based on passing all required values
    public ReefPolygon(List<ReefPolygonPoint> reefPolygonPointList ){
```

```
        if ( !reefPolygonPointList.isEmpty() ){

            this._reefId = reefPolygonPointList.get( 0 ).getReefId();

            List<LatLng> latLngList = new ArrayList<>();

            for ( ReefPolygonPoint reefPolygonPoint : reefPolygonPointList ){

                latLngList.add( new LatLng( reefPolygonPoint.getReefPolygonPointLatitude(),
reefPolygonPoint.getReefPolygonPointLongitude() ) );

            }

            this._reefPolygonPointLatLngs = latLngList;

        }

    }

    //
    // GETTERS
    //

    // We provide individual public getter functions so that pieces of data can be read from
    // each ReefPolygon object.

    // Getting id
    public int getReefId(){
        return this._reefId;
    }

    // Getting reef polygon
    public List<LatLng> getReefPolygonPoints(){
        return this._reefPolygonPointLatLngs;
    }

}
```

## B.22 ReefPolygonPoint.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types;

import android.database.Cursor;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.ReefPolygonsEntry;

/**
 * Created by fle125 on 31/03/2017.
 */
```

```
public class ReefPolygonPoint {

    // Private Variables
    // These are the components intrinsic to a ReefPolygonPoint, rather than values that should be
    // derived from the Reef.

    private int _reefPolygonsId;
    private int _reefPolygonPointOrder;
    private double _reefPolygonPointLatitude;
    private double _reefPolygonPointLongitude;
    private int _reefId;

    //
    // CONSTRUCTORS
    //

    // Empty constructor
    public ReefPolygonPoint(){
    }

    // Constructor based on passing all required values
    public ReefPolygonPoint(int reefPolygonsId, int reefPolygonPointOrder, double reefPolygonPointLatitude,
double reefPolygonPointLongitude, int reefId ){

        this._reefPolygonsId = reefPolygonsId;
        this._reefPolygonPointOrder = reefPolygonPointOrder;
        this._reefPolygonPointLatitude = reefPolygonPointLatitude;
        this._reefPolygonPointLongitude = reefPolygonPointLongitude;
        this._reefId = reefId;

    }

    // Constructor based on passing a single Cursor to a SiteEntry
    public ReefPolygonPoint(Cursor cursor) {

        this._reefPolygonsId =
cursor.getInt(cursor.getColumnIndex(ReefPolygonsEntry.REEF_POLYGONS_TABLE_COLUMN_ID));
        this._reefPolygonPointOrder =
cursor.getInt(cursor.getColumnIndex(ReefPolygonsEntry.REEF_POLYGONS_TABLE_COLUMN_POINT_ORDER));
        this._reefPolygonPointLatitude =
cursor.getDouble(cursor.getColumnIndex(ReefPolygonsEntry.REEF_POLYGONS_TABLE_COLUMN_POINT_LATITUDE));
        this._reefPolygonPointLongitude =
cursor.getDouble(cursor.getColumnIndex(ReefPolygonsEntry.REEF_POLYGONS_TABLE_COLUMN_POINT_LONGITUDE));
        this._reefId =
cursor.getInt(cursor.getColumnIndex(ReefPolygonsEntry.REEF_POLYGONS_TABLE_COLUMN_REEF_ID));

    }

    //
    // GETTERS
    //
```

```
    // We provide individual public getter functions so that pieces of data can be read from
    // each ReefPolygonPoint object.

    // Getting id
    public int getReefPolygonsId(){
        return this._reefPolygonsId;
    }

    // Getting reefName
    public int getReefPolygonPointOrder(){
        return this._reefPolygonPointOrder;
    }

    // Getting reefId
    public double getReefPolygonPointLatitude(){
        return this._reefPolygonPointLatitude;
    }

    // Getting latitude
    public double getReefPolygonPointLongitude(){ return this._reefPolygonPointLongitude; }

    // Getting longitude
    public int getReefId(){ return this._reefId; }

}
```

## B.23 Voyage.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types;

import android.database.Cursor;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.VoyageEntry;

import static java.lang.Math.random;

/**
 * Created by fle125 on 31/03/2017.
 */

public class Voyage {

    // Private Variables
    // These are the components intrinsic to a Voyage, rather than simply a collection of Dives
    // For instance - the Voyage startDate and stopDate may be different than the first or last
    // date of Dives if weather was bad, etc.
    // On the other hand, it is not possible for the number of CoTS removed on a Voyage to be
    // different to the sum removed during the constituent Dives. We therefore do not want to
    // record the number of CoTS removed during a Voyage in the object itself, but we do want to
    // provide a method that returns the number of CoTS removed during the voyage by summing up
```

```
    // the number removed in each dive.


    private int _voyageId;
    private int _voyageVoyageNumber;
    private String _voyageStartDate;
    private String _voyageStopDate;
    private int _vesselId;


    //
    // CONSTRUCTORS
    //


    // Empty constructor
    public Voyage(){


    }


    // Constructor
    public Voyage(int voyageId, int voyageVoyageNumber, String voyageStartDate, String voyageStopDate, int
vesselId){
        this._voyageId = voyageId;
        this._voyageVoyageNumber = voyageVoyageNumber;
        this._voyageStartDate = voyageStartDate;
        this._voyageStopDate = voyageStopDate;
        this._vesselId = vesselId;
    }


    // Constructor
    public Voyage(Cursor cursor) {


        this._voyageId = cursor.getInt(cursor.getColumnIndex(VoyageEntry.VOYAGE_TABLE_COLUMN_ID));
        this._voyageVoyageNumber =
cursor.getInt(cursor.getColumnIndex(VoyageEntry.VOYAGE_TABLE_COLUMN_VOYAGE_NUMBER));
        this._voyageStartDate =
cursor.getString(cursor.getColumnIndex(VoyageEntry.VOYAGE_TABLE_COLUMN_START_DATE));
        this._voyageStopDate =
cursor.getString(cursor.getColumnIndex(VoyageEntry.VOYAGE_TABLE_COLUMN_STOP_DATE));
        this._vesselId = cursor.getInt(cursor.getColumnIndex(VoyageEntry.VOYAGE_TABLE_COLUMN_VESSEL_ID));


    }


    //
    // GETTERS
    //


    // We provide individual public getter functions so that pieces of data can be read from
    // each Voyage object.


    // Getting vesselVoyage
    public int getVoyageId(){ return this._voyageId; }


    // Getting vesselVoyage
```

```
    public int getVoyageNumber(){ return this._voyageVoyageNumber; }

    // Getting startDate
    public String getStartDate(){
        return this._voyageStartDate;
    }

    // Getting stopDate
    public String getStopDate(){
        return this._voyageStopDate;
    }

    // Getting vessel
    public int getVesselId(){
        return this._vesselId;
    }

    //
    // DERIVED VALUES
    //

    // We also provide functions for derived values, like the total number of CoTS removed
    // and the catch per unit effort

    public double meanLatitude() {

            return -16.9186 + random(); /* Cairns Latitude */

    }

    public double meanLongitude() {

            return 145.7781 + random(); /* Cairns Longitude */

    }

}
```

## B.24 Vessel.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types;

import android.database.Cursor;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract;

/**
 * Created by fle125 on 31/03/2017.
 */
```

```
public class Vessel {

    // Private Variables
    // These are the components intrinsic to a Vessel, rather than values that should be derived
    // from the Voyages on which a Vessel was involved.

    private int _vesselId;
    private String _vesselName;
    private String _vesselShortName;

    //
    // CONSTRUCTORS
    //

    // Empty constructor
    public Vessel(){
    }

    // Constructor based on passing all required values
    public Vessel(int vesselId, String vesselName, String vesselShortName ){

        this._vesselId = vesselId;
        this._vesselName = vesselName;
        this._vesselShortName = vesselShortName;
    }

    // Constructor based on passing a single Cursor to a SiteEntry
    public Vessel(Cursor cursor) {


        this._vesselId =
cursor.getInt(cursor.getColumnIndex(COTSDataContract.VesselEntry.VESSEL_TABLE_COLUMN_ID));
        this._vesselName =
cursor.getString(cursor.getColumnIndex(COTSDataContract.VesselEntry.VESSEL_TABLE_COLUMN_VESSEL_NAME));
        this._vesselShortName =
cursor.getString(cursor.getColumnIndex(COTSDataContract.VesselEntry.VESSEL_TABLE_COLUMN_VESSEL_SHORT_NAME))
;


    }


    //
    // GETTERS
    //

    // We provide individual public getter functions so that pieces of data can be read from
    // each Vessel object.

    // Getting id
    public int getVesselId(){
        return this._vesselId;
    }

    // Getting vesselName
```

```
    public String getVesselName(){
        return this._vesselName;
    }

    // Getting vesselShortName
    public String getVesselShortName(){
        return this._vesselShortName;
    }

}
```

## B.25 Rhis.java

```
package au.csiro.cotscontrolcentre_decisionsupporttool_0_0.types;

import android.database.Cursor;

import au.csiro.cotscontrolcentre_decisionsupporttool_0_0.data.COTSDataContract.RhisEntry;

/**
 * Created by fle125 on 31/03/2017.
 */

public class Rhis {

    // Private Variables
    // These are the components intrinsic to a Rhis, rather than a nearby Dive or Site or Reef where
    // the Rhis took place

    private int _rhisId;
    private String _rhisDate;
    private double _rhisCoralCover;
    private int _rhisCOTSAdult;
    private int _rhisCOTSJuvenile;
    private String _rhisVisibility;
    private int _siteId;

    //
    // CONSTRUCTORS
    //

    // Empty constructor
    public Rhis(){

    }

    // Constructor
    public Rhis(int rhisId, String rhisDate, double rhisCoralCover, int rhisCOTSAdult, int
rhisCOTSJuvenile, String rhisVisibility, int siteId ){
```

```
        this._rhisId = rhisId;
        this._rhisDate = rhisDate;
        this._rhisCoralCover = rhisCoralCover;
        this._rhisCOTSAdult = rhisCOTSAdult;
        this._rhisCOTSJuvenile = rhisCOTSJuvenile;
        this._rhisVisibility = rhisVisibility;
        this._siteId = siteId;
    }

    // Constructor
    public Rhis(Cursor cursor) {
        this._rhisId = cursor.getInt(cursor.getColumnIndex(RhisEntry.RHIS_TABLE_COLUMN_ID));
        this._rhisDate = cursor.getString(cursor.getColumnIndex(RhisEntry.RHIS_TABLE_COLUMN_DATE));
        this._rhisCoralCover =
cursor.getDouble(cursor.getColumnIndex(RhisEntry.RHIS_TABLE_COLUMN_AVERAGE_CORAL_COVER));
        this._rhisCOTSAdult =
cursor.getInt(cursor.getColumnIndex(RhisEntry.RHIS_TABLE_COLUMN_COTS_ADULTS));
        this._rhisCOTSJuvenile =
cursor.getInt(cursor.getColumnIndex(RhisEntry.RHIS_TABLE_COLUMN_COTS_JUVENILES));
        this._rhisVisibility =
cursor.getString(cursor.getColumnIndex(RhisEntry.RHIS_TABLE_COLUMN_VISIBILITY));
        this._siteId = cursor.getInt(cursor.getColumnIndex(RhisEntry.RHIS_TABLE_COLUMN_SITE_ID));

    }

    //
    // SETTERS
    //

    // We do not provide any public setter functions because all data should be created
    // together, not edited piecemeal.

    //
    // GETTERS
    //

    // We provide individual public getter functions so that pieces of data can be read from
    // each object.

    // Getting id
    public int getRhisId(){
        return this._rhisId;
    }

    // Getting date
    public String getRhisDate(){
        return this._rhisDate;
    }

    // Getting averageCoralCover
    public double getRhisCoralCover(){ return this._rhisCoralCover; }
```

```
    // Getting waterTemperature
    public int getRhisCOTSAdults(){
        return this._rhisCOTSAdult;
    }

    // Getting waterTemperature
    public int getRhisCOTSJuveniles(){
        return this._rhisCOTSJuvenile;
    }

    // Getting visibility
    public String getRhisVisibility(){
        return this._rhisVisibility;
    }

    // Getting siteId
    public Integer getSiteId(){
        return this._siteId;
    }

    //
    // DERIVED VALUES
    //

    // Although at the moment we do not provide derived values from these objects, in future we
    // are likely to.

    //
    // COMPARATORS
    //

    // Although at the moment we do not provide compartors for these objects, in future we
    // are likely to.

}
```

## B.26 main_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>

<!-- No styling information should be entered in this file, just layouts and ids -->
<!-- All style information is stored in /raw/styles.xml -->

<LinearLayout style="@style/AppTheme.mainLinearLayoutVertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="au.csiro.cotscontrolcentre_decisionsupporttool_0_0.MainActivity">

    <FrameLayout
```

```
        android:id="@+id/display_info_fragment_container_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:visibility="gone">
    </FrameLayout>

    <FrameLayout
        android:id="@+id/display_map_fragment_container_view"
        android:layout_width="0dp"
        android:layout_weight="0.75"
        android:layout_height="match_parent">
    </FrameLayout>

</LinearLayout>
```

## B.27 display_map.xml

```
<?xml version="1.0" encoding="utf-8"?>

<!-- No styling information should be entered in this file, just layouts and ids -->
<!-- All style information is stored in /raw/styles.xml -->

<LinearLayout style="@style/AppTheme.mainLinearLayoutVertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="au.csiro.cotscontrolcentre_decisionsupporttool_0_0.MainActivity">

    <FrameLayout
        android:id="@+id/map_frame"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="0.75"
        android:visibility="visible">

        <fragment xmlns:android="http://schemas.android.com/apk/res/android"
            android:id="@+id/map_main"
            android:name="com.google.android.gms.maps.SupportMapFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:clickable="true"
            android:focusable="true"/>

        <LinearLayout style="@style/AppTheme.displayButtonRow">

            <Button style="@style/AppTheme.displayButton"
                android:id="@+id/mantaButton"
                android:text="Manta tows" />

            <Button style="@style/AppTheme.displayButton"
```

```
                android:id="@+id/cullButton"
                android:text="Cull density" />

        </LinearLayout>

        <RelativeLayout
            android:id="@+id/loadingPanel"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="center"
            android:visibility="gone">

            <ProgressBar
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:indeterminate="true" />
        </RelativeLayout>

    </FrameLayout>

</LinearLayout>
```

## B.28 display_info.xml

```
<?xml version="1.0" encoding="utf-8"?>

<!-- No styling information should be entered in this file, just layouts and ids -->
<!-- All style information is stored in /raw/styles.xml -->

<LinearLayout style="@style/AppTheme.mainLinearLayoutVertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="au.csiro.cotscontrolcentre_decisionsupporttool_0_0.MainActivity">

    <FrameLayout
        android:id="@+id/info_frame"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="0.75"
        android:visibility="visible">

        <LinearLayout style="@style/AppTheme.overlayPanel"
            android:id="@+id/reef_info_overlay">

            <TextView style="@style/AppTheme.overlayTextHeading"
                android:id="@+id/reef_info_overlay_reef_name" />

            <LinearLayout style="@style/AppTheme.overlayInfoPanelInfo"
                android:id="@+id/overlay_info_panel">
```

```
        <TextView style="@style/AppTheme.overlayTextBody"
            android:id="@+id/reef_info_overlay_reef_reef_mode" />

        <TextView style="@style/AppTheme.overlayTextBody"
            android:id="@+id/reef_info_overlay_reef_number_of_sites" />

        <TextView style="@style/AppTheme.overlayTextBody"
            android:id="@+id/reef_info_overlay_last_cull_date" />

        <TextView style="@style/AppTheme.overlayTextBody"
            android:id="@+id/reef_info_overlay_last_manta_date" />

        <TextView style="@style/AppTheme.overlayTextBody"
            android:id="@+id/reef_info_overlay_number_of_days_since_last_manta" />

        <TextView style="@style/AppTheme.overlayTextBody"
            android:id="@+id/reef_info_overlay_manta_due" />

        <TextView style="@style/AppTheme.overlayTextBody"
            android:id="@+id/reef_info_overlay_reef_total_cots_culled_during_last_cull" />

        <TextView style="@style/AppTheme.overlayTextBody"
            android:id="@+id/reef_info_overlay_reef_total_cots_seen_during_last_manta" />

        <TextView style="@style/AppTheme.overlayTextBody"
            android:id="@+id/reef_info_overlay_reef_any_manta_scars_seen_during_last_manta" />

        <TextView style="@style/AppTheme.overlayTextBody"
            android:id="@+id/reef_info_overlay_reef_number_of_sites_with_manta_cots_or_scars" />

    </LinearLayout>

    <LinearLayout style="@style/AppTheme.overlayInfoPanelWorkplan"
        android:id="@+id/overlay_workplan_panel">

        <TextView style="@style/AppTheme.overlayTextBodyHeading"
            android:id="@+id/reef_info_overlay_workplan_heading"
            android:text="Recommended Workplan \n"/>

        <TextView style="@style/AppTheme.overlayTextBody"
            android:id="@+id/reef_info_overlay_workplan" />

    </LinearLayout>


    <LinearLayout style="@style/AppTheme.actionButtonPanel"
        android:id="@+id/action_button_panel">

        <Button style="@style/AppTheme.actionButton"
```

```
                    android:id="@+id/loadSurveillanceFileButton"
                    android:text="Load new data ..." />

                <Button style="@style/AppTheme.actionButton"
                    android:id="@+id/generateWorkplanButton"
                    android:text="Generate Workplan" />

            </LinearLayout>

        </LinearLayout>

    </FrameLayout>

</LinearLayout>
```

## B.29 site_marker_info.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/site_marker_info"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:background="@drawable/circle"
    android:gravity="center_vertical">

    <TextView
        android:id="@+id/site_marker_info_text"
        android:textSize="35px"
        android:textColor="@color/colorWhite"
        android:text="1"
        android:textAlignment="center"
        android:layout_width="50px"
        android:layout_height="50px"/>

</LinearLayout>
```

## B.30 styles.xml

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Set up default colors -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
        <!--<item name="@android:windowBackground">@drawable/default_background</item>-->
    </style>
```

```
<!-- Set up splash screen -->
<style name="AppTheme.splashScreen">
    <!--<item name="@android:windowBackground">@drawable/splash_screen</item>-->
</style>

<style name="AppTheme.mainLinearLayoutVertical">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">match_parent</item>
    <item name="android:orientation">horizontal</item>
    <item name="android:animateLayoutChanges">true</item>
    <item name="android:background">@color/colorWhite</item>
</style>

<style name="AppTheme.overlayPanel">
    <item name="android:layout_width">400px</item>
    <item name="android:layout_height">match_parent</item>
    <item name="android:layout_margin">25px</item>
    <item name="android:layout_gravity">right</item>
    <item name="android:visibility">visible</item>
    <item name="android:orientation">vertical</item>
    <item name="android:background">@color/colorWhite</item>
</style>

<style name="AppTheme.overlayInfoPanel">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">fill_parent</item>
    <item name="android:layout_weight">1</item>
    <item name="android:orientation">vertical</item>
    <item name="android:background">@color/colorWhite</item>
</style>

<style name="AppTheme.overlayInfoPanelInfo" parent="AppTheme.overlayInfoPanel">
    <item name="android:visibility">visible</item>
</style>

<style name="AppTheme.overlayInfoPanelWorkplan" parent="AppTheme.overlayInfoPanel">
    <item name="android:visibility">gone</item>
</style>

<style name="AppTheme.overlayText">
    <item name="android:layout_height">wrap_content</item>
    <item name="android:layout_width">wrap_content</item>
    <item name="android:background">@color/colorWhite</item>
    <item name="android:textColor">@color/colorBlack</item>
</style>

<style name="AppTheme.overlayTextHeading" parent="AppTheme.overlayText">
    <item name="android:textSize">30px</item>
    <item name="android:layout_margin">20px</item>
    <item name="android:textStyle">bold</item>
```

```
    <item name="android:textColor">@color/colorPrimary</item>
</style>

<style name="AppTheme.overlayTextBodyHeading" parent="AppTheme.overlayText">
    <item name="android:textSize">26px</item>
    <item name="android:layout_marginLeft">20px</item>
    <item name="android:layout_marginRight">20px</item>
    <item name="android:layout_marginTop">5px</item>
    <item name="android:layout_marginBottom">5px</item>
</style>

<style name="AppTheme.overlayTextBody" parent="AppTheme.overlayText">
    <item name="android:textSize">20px</item>
    <item name="android:layout_marginLeft">20px</item>
    <item name="android:layout_marginRight">20px</item>
    <item name="android:layout_marginTop">5px</item>
    <item name="android:layout_marginBottom">5px</item>
</style>

<style name="AppTheme.displayButtonRow">
    <item name="android:layout_width">wrap_content</item>
    <item name="android:layout_height">75px</item>
    <item name="android:orientation">horizontal</item>
    <item name="android:layout_gravity">right</item>
    <item name="android:layout_marginTop">25px</item>
    <item name="android:layout_marginRight">25px</item>
</style>

<style name="AppTheme.displayButton">
    <item name="android:layout_width">200px</item>
    <item name="android:layout_height">match_parent</item>
    <item name="android:visibility">visible</item>
    <item name="android:background">@color/colorWhite</item>
    <item name="android:textColor">@color/colorPrimary</item>
    <item name="android:textStyle">bold</item>
    <item name="android:layout_marginLeft">25px</item>
</style>

<style name="AppTheme.actionButtonPanel">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:orientation">vertical</item>
    <item name="android:layout_gravity">bottom</item>
</style>

<style name="AppTheme.actionButton">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:visibility">visible</item>
    <item name="android:background">@color/colorPrimary</item>
```

```
        <item name="android:textColor">@color/colorWhite</item>
        <item name="android:textStyle">bold</item>
        <item name="android:layout_marginTop">25px</item>
    </style>
</resources>
```

## B.31 colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- CSIRO Core Colour Midday Blue R0 G169 B206 -->
    <color name="colorPrimary">#00a9ce</color>
    <!-- CSIRO Core Colour Sky Blue R65 G182 B230 -->
    <color name="colorPrimaryLight">#41b6e6</color>
    <!-- CSIRO Core Colour Midnight Blue R0 G49 B60 -->
    <color name="colorPrimaryDark">#00313c</color>
    <!-- CSIRO Core Colour Sky Blue R65 G182 B230 with ~10% opacity -->
    <color name="colorPrimaryVeryVeryVeryLight">#1a41b6e6</color>
    <!-- CSIRO Core Colour Sky Blue R65 G182 B230 with ~20% opacity -->
    <color name="colorPrimaryVeryVeryLight">#3341b6e6</color>
    <!-- CSIRO Core Colour Sky Blue R65 G182 B230 with ~30% opacity -->
    <color name="colorPrimaryVeryLight">#4C41b6e6</color>
    <!-- CSIRO Accent Colour Orange -->
    <color name="colorAccent">#e87722</color>
    <!-- Generic Colours -->
    <color name="colorWhite">@android:color/white</color>
    <color name="colorBlack">@android:color/black</color>
    <color name="colorGray">@android:color/darker_gray</color>
    <color name="colorGrayTransparent">#aaaaaaaa</color>
</resources>
```

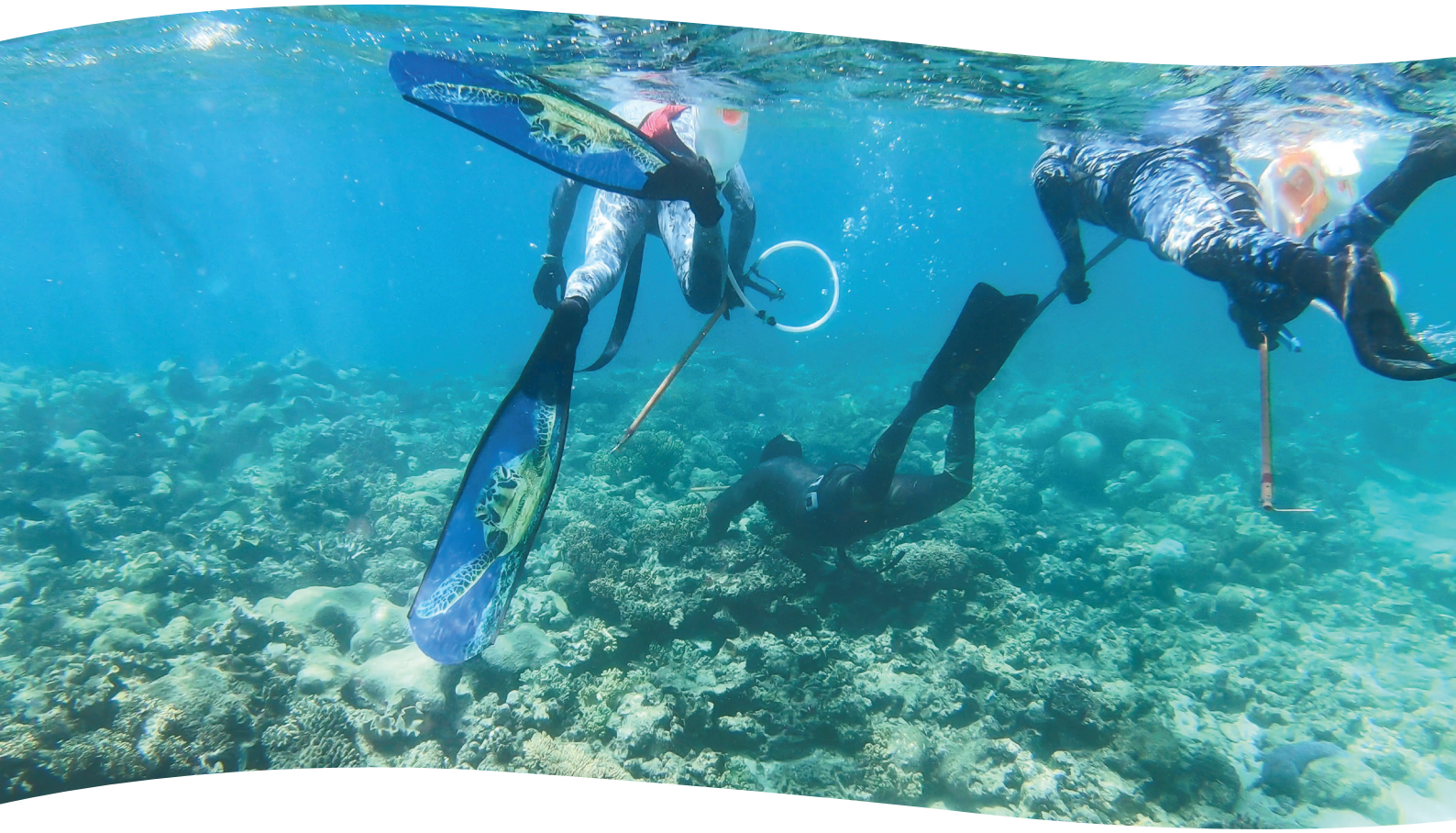
## B.32 strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE resources [
    <!ENTITY major_version "0">
    <!ENTITY minor_version "0">
    ]>


<resources>
    <!-- Application title and version numbers -->
    <string name="app_name">COTS Control Centre - Decision Support Tool Version
&major_version;.&minor_version;</string>
    <string name="app_major_version_number">&major_version;</string>
    <string name="app_minor_version_number">&minor_version;</string>


    <!-- Content authority -->
    <string
name="content_authority">au.csiro.cotscontrolcentre_decisionsupporttool_&major_version;_&minor_version;</st
ring>
```

```
</resources>
```

## B.33 circle.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
       android:shape="oval">
    <solid android:color="@color/colorPrimaryLight" />

</shape>
```



Reef & Rainforest
RESEARCH CENTRE

www.nesptropical.edu.au