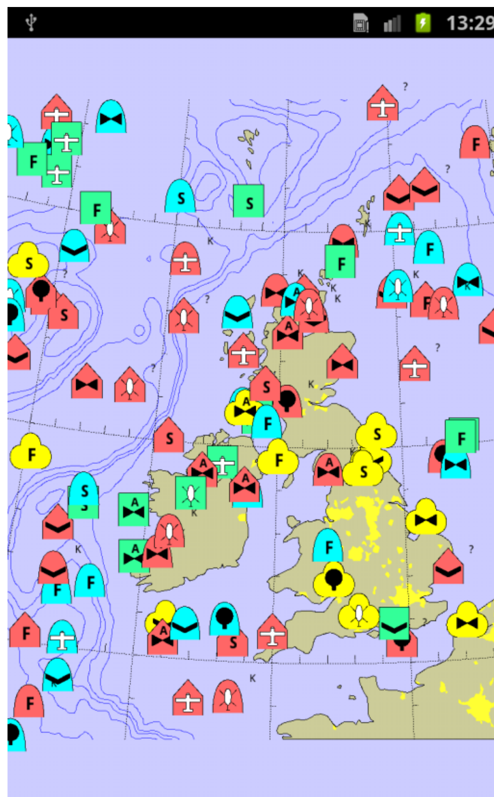
MapLink Pro for Android™

Developer's Guide

AUM1114|21 January 2022| Status: Approved



© Envitia Ltd. 2022

North Heath Lane, Horsham, West Sussex, RH12 5UX, United Kingdom
Tel: +44 1403 273 173 Email: info@envitia.com
www.envitia.com

TABLE OF CONTENTS

1 INTRODUCTION 6
1.1 Training, Consultancy and Sub-Contracting 6
1.2 Trademarks 6
1.3 Glossary 6
2 INTRODUCTION TO MAPLINK PRO FOR ANDROID 7
2.1 CoreSDK 7
2.2 TerrainSDK 7
2.3 GeoPackageSDK 8
2.4 OWSContextSDK 8
2.5 DirectImportSDK 8
2.5.1 Supported Data Formats 8
2.5.2 Data Layout and scale bands 9
2.5.3 Data Processing and Display 9
2.5.4 Callbacks 9
2.5.5 Vector specific settings and styling 10
2.5.6 Raster specific settings 10
2.5.7 Caching 11
2.5.7.1 In-Memory Cache 11
2.5.7.2 On Disk Cache 11
2.5.7.3 Raster Draw Cache 11
2.5.8 Optimising Raster Data for Direct Import 12
2.5.8.1 Creating Overview Layers 12
2.5.8.2 Combining Raster Mosaics 12
2.6 Application Architecture 13
2.7 The View Model 13
2.8 Error Handling 14
2.9 View and Interaction Modes 14
2.10 Coordinates and Positions 14
2.11 Configuration Data 15
3 INSTALLATION 16
3.1 Prerequisites 16
3.2 Requirements 16
3.2.1 Supported Architectures 16
3.3 Installation Layout 17
3.4 Getting started 17
4 INCLUDING MAPLINK PRO FOR ANDROID WHEN BUILDING YOUR APPLICATION 18
4.1 Importing the MapLink CoreSDK into an Android Studio Project 18
4.1.1 Why Aren't the Native Libraries Inside the Jar File? 18
4.2 Additional MapLink SDKs 18
4.2.1 WMTS Data Layer 18
4.2.2 CADRG Data Layer 19
4.2.3 Terrain SDK 19
4.2.4 GeoPackage SDK 19
4.2.5 OWS Context SDK 19
4.2.6 Direct Import SDK 19
4.3 Importing JavaDoc for MapLink into Android Studio 19
5 MAPLINK COMPONENTS AND CONCEPTS 21
5.1 Things Every Application Must Do 21
5.1.1 Load the Native Libraries 21
5.1.2 Initialise the MapLink Environment 21
5.1.3 Load the Standard Configuration 22
5.1.4 Unlock MapLink Components 22
5.2 The Drawing Surface 23
5.3 Data Layers 23

© 2022 Envitia Ltd

5.3.1 Background Layers 23
5.3.1.1 Web Map Service DataLayers....24
5.3.2 Overlay Layers 24
5.4 Java enums....24
5.5 2D Vector Geometry 25
5.5.1 TSLEntity....25
5.5.2 TSLPolyline 25
5.5.3 TSLPolygon 26
5.5.4 TSLText....26
5.5.5 TSLSymbol 27
5.5.6 TSLEllipse....27
5.5.7 TSLArc 28
5.5.8 TSLRectangle 28
5.5.9 TSLEntitySet and other Collections....28
5.6 Rendering Configuration 29
5.6.1 Entity Based Rendering 29
5.6.2 Feature Based Rendering....29
5.6.3 Determining the Source of Rendering Attributes 29
5.7 MapLink Rendering Attributes....30
5.7.1 Generic Attributes....30
5.7.2 Line Rendering Attributes....30
5.7.3 Area Rendering Attributes....31
5.7.4 Text Rendering Attributes 31
5.7.5 Symbol Rendering Attributes....33
5.7.6 Raster Icon Symbols 34
5.7.7 Minimum Attribute Requirements....34
5.7.8 Why Can't I See My Object?....35
5.8 Decluttering 36
5.8.1 Declutter Feature Name and ID 36
5.8.2 Declutter Status 36
5.8.3 Automatic Decluttering on Zoom....37
5.8.4 Declutter of Raster Features in Maps 37
5.9 Searching Your Data 37
5.9.1 Finding the Entity at a Point on the Screen....37
5.9.2 Finding All Entities Within an Area....38
5.9.3 Finding data within a TSLCustomDataLayer....39
5.10 Terrain SDK 41
5.10.1 Queries 41
5.11 Direct Import SDK 43

DEPLOYMENT OF AN APPLICATION 44

5.12 Loading MapLink Configuration Files....44
5.12.1 Loading MapLink Configuration Files from the Assets Directory 44
5.12.2 Loading MapLink Configuration Files from Internal Storage 45
5.12.3 Removing unnecessary configuration data 46
5.13 Loading Other MapLink Files From the Asset Directory....46
5.14 MapLink Application Permissions....47
5.15 MapLink Application Feature Declarations 47
5.16 Proguard 47
5.17 Licencing 47

6 MAPLINK AND OPENGL 48

6.1 Handling Power Events....48

7 THREADING 49

7.1 TSLEGLSurfaceView Rendering Thread 49
7.1.1 OpenGL and Threads....49

8 CONFIGURATION DATA FORMATS.... 50

8.1 Fonts 50

9 MAPLINK BUILD AND ARCHITECTURE NOTES 51
© 2022 Envitia Ltd

10 SAMPLE APPLICATION WALKTHROUGH - BASIC MAPLINK APPLICATION ... 52

10.1 Creating the Project 52
10.2 Linking against MapLink for Android 53
10.2.1 Java Libraries 53
10.2.2 Native Libraries 54
10.3 Loading the Native Libraries 55
10.4 Load the MapLink Standard Configuration 55
10.5 Unlocking licenced components 56
10.6 Creating the Activity's User Interface 56
10.7 Add a Map Layer 58
10.8 Add the Map Data Layer to the Surface View 58
10.9 Adding Interaction Modes 59
10.10 Launching the application 60

11 BASIC MAPLINK APPLICATION WITH APP6A SYMBOLS 61

11.1 Loading the APP6A Symbols 61
11.2 Continuous Rendering on TSLEGLSurfaceView 61
11.3 Buffered Data Layers 62
11.4 Entity Storage Strategy 62
11.5 Implementing the Tracks Data Layer 62
11.5.1 Creating APP6A Symbols 63
11.5.2 Drawing the symbols 63
11.5.3 Releasing Entity Resources 64

TABLE OF FIGURES

Figure 1 Polyline 25
Figure 2 Polygon 26
Figure 3 Text 26
Figure 4 Symbols 27
Figure 5 Ellipse 27
Figure 6 Arc 28
Figure 7 Rectangle 28

© 2022 Envitia Ltd

1 INTRODUCTION

This document a guide for developers to designing and implement solutions using MapLink Pro for Android.

1.1 Training, Consultancy and Sub-Contracting

Envitia provides a range of training options to help you get the best from MapLink Pro and MapLink Studio. These courses greatly help to accelerate your development, produce optimised applications more quickly and to explore alternative ways of achieving your objectives.

Dedicated consultancy can also be provided either on site or remotely, allowing our experienced developers to guide you towards the most appropriate approach to your application arena. Customers frequently find this useful when adding additional new functionality to their systems.

Envitia can also help accelerate your development by developing the MapLink component of your application for you or by undertaking a more extensive part of your project for you. Envitia has extensive experience of developing applications internally and for external customers.

If you wish to discuss these opportunities, please contact Sales by email sales@envitia.com or by phone: +44 1403 273173.

1.2 Trademarks

Android is a trademark of Google Inc.

1.3 Glossary

API	Application Programming Interface
DMS	Digital Mapping System
DPI	Dots per Inch
EPSG	European Petroleum Survey Group. This organisation defines a standardised database of Coordinate Systems. These contain numeric codes associated with coordinate system definitions http://www.epsg.org/
IDE	Integrated Development Environment
JPEG	JPEG raster format
Layer	A container that represents a collection of Geometry be it a Map or an Overlay.
STL	C++ Standard Template Library
SDK	Software Developers Kit
TMF	Envitia Map Format. Native geometry file format.
TIFF	TIFF raster format
TMC	The units that MapLink Pro uses to define a rectilinear coordinate space for drawing Map data and Overlay data with.

2 INTRODUCTION TO MAPLINK PRO FOR ANDROID

MapLink Pro for Android is built on the C++ MapLink Pro API. Therefore, the Java API for MapLink Pro reflects many of the capabilities and concepts of the MapLink C++ runtime with a familiar MapLink API designed for Java.

Because the capabilities and concepts are very similar, we would recommend referring to the 'MapLink Pro Developer's Guide' in addition to this document and the Javadoc documentation.

2.1 CoreSDK

The Core SDK is the basis of all MapLink Pro applications. Like all MapLink SDKs, it is modular and flexible. Unlike many other products, MapLink does not dictate the architecture of your application. It is flexible enough to be easily integrated into whatever architecture best fits your application domain.

At its simplest level the MapLink Core Android SDK can be summarised in three concepts and two sentences:

- Data is loaded into layers
- Layers are displayed on drawing surfaces

At a more complex level the Core SDK provides the following basic facilities to a MapLink application:

- Visualisation of vector maps and overlays
- Visualisation of raster maps and overlays
- Loading of vector and raster data
- A suite of vector geometric primitives
- Access to attributes stored within maps generated by MapLink Studio
- Layering and decluttering of features
- Access to the powerful coordinate system engine for map transformations
- Multi-threaded map display for smooth, responsive applications

2.2 TerrainSDK

The Terrain SDK provides fast access to layered terrain data that has been prepared through MapLink Studio and directly loaded DTED/DMED data.

Height data may be queried on a spot, line or height basis for use by the application directly, or with the `TSLTerrainViewShed` class, which provides built-in line of sight and view shed analysis.

The Java API includes the full functionality of the `TSLTerrainDatabase`, `TSLDTEDTerrainDatabase`, and `TSLTerrainViewShed` classes, but does not include any of the contouring functionality.

For more information on the TerrainSDK see section 2.2 or the 'MapLink Pro Developer's Guide'.

2.3 GeoPackageSDK

The GeoPackage SDK allows the user to read, analyse and display data from GeoPackage data files.

A GeoPackage is a platform independent SQLite database schema for storing and transferring geographic vector features and image tiles. The schema contains specified definitions, integrity assertions, format limitations and content constraints.

A GeoPackage may be 'empty' (contain user data table(s) for vector features and/or tile matrix pyramids with no row record content) or contain one or many vector feature type records and/or one or many tile matrix pyramid tile images. GeoPackage metadata can describe GeoPackage data contents and identify external data synchronisation sources and targets. A GeoPackage may contain spatial indexes on feature geometries.

2.4 OWSContextSDK

The OWSContext SDK allows a user to read, analyse and display OWSContext documents within MapLink.

This SDK can read several offering types from an OWSContext document and provides a plugin interface to allow other offering types to be integrated with the SDK:

- GML
- WMS
- WMTS

2.5 DirectImportSDK

The Direct Import SDK allows an application to load a wide variety of data formats at runtime in a scalable and performant manner.

The `TSLDirectImportDataLayer` can load both vector and raster data, including mixed raster and vector from a single file. The layer provides the ability to re-project data to the specified output coordinate system along with various vector and raster processing options.

Many of the options and concepts used by the Direct Import layer are similar to those in MapLink Studio.

This includes the ability to export a feature rendering configuration from MapLink Studio in order to style vector data within the Direct Import layer.

2.5.1 Supported Data Formats

The `TSLDirectImportDataLayer` does not impose any restrictions on file formats. Instead these are determined by the available implementations of `TSLDirectImportDriver`.

Each `TSLDirectImportDriver` may support a range of configuration options. These options may be set globally via the configuration files under the MapLink `config directory/directimport`.

2.5.2 Data Layout and scale bands

The `TSLDirectImportDataLayer` may load a mixture of raster and vector data, which may be displayed in any order.

One data path (A file path, web service URL or other data identifier) may correspond to multiple instances of `TSLDirectImportDataSet`, with each data set corresponding to a sub-layer within the data. Simple formats such as shapefiles will only contain a single dataset, which will correspond to the vector feature within the data. These data sets are handled independently of each other, and as such may be loaded on a selective basis. Data sets may also be loaded with different per-dataset settings such as feature rendering, and raster adjustments.

To load a data set, the application must call `addScaleBand` at least once. Each scale band within the data layer functions in a similar way to detail layers in a map, or layers within a MapLink Studio project.

Only one scale band will be displayed by the data layer at a time.

The selection of scale bands is based upon a calculated display scale, such as 1:100,000. For this to be accurate, the application should set the parameters of the display via `TSLDrawingSurfaceBase::setDeviceCapabilities`. On some platforms these capabilities may be set automatically by the drawing surface.

A data set may be loaded into multiple scale bands. This may be used to display data as a background for all display scales. For raster data, overview datasets may be loaded if present in the original data. These are reduced resolution versions of the data set suitable for loading into overview layers.

Data loaded into a scale band will be split into tiles for processing/display. These tiling levels may either be set by the application or calculated automatically. The automatic tiling calculation is based upon the minimum display scale of the band and will create more tiles for more detailed scales. Applications must ensure that data is loaded at an appropriate scale to maintain performance.

2.5.3 Data Processing and Display

When a data set is loaded into the layer it will be split into a number of tiles (based on the scale band configuration) and processed asynchronously. Once a tile has been processed it will be stored in the on-disk cache and displayed. If the data needs to be reloaded after this point it will be loaded from the on-disk cache.

Data will be scheduled for loading based on the current view extent and the `extentExpansion` setting of the layer. The application may also request that a specific extent be processed by calling `preprocessData`.

Complex vector data or large amounts of raster data may take a long time to process. It is advisable to call `preprocessData` for these datasets prior to the point they need to be displayed to pre-process the data into the on-disk cache.

2.5.4 Callbacks

The `TSLDirectImportDataLayer` is fully asynchronous and will rarely block the calling thread for any significant amount of time.

To achieve this, the following callback classes are provided:

- `TSLDirectImportDataLayerCallbacks` - The application should always provide an implementation of this class. It provides the application with feedback on data processing and is used to request that the application redraws the drawing surface.
- `TSLDirectImportDataLayerAnalysisCallbacks` - The application should provide an implementation of this class when performing data analysis operations. An implementation of this class is not required when loading data for display.

© 2022 Envitia Ltd

2.5.5 Vector specific settings and styling

Other than styling/feature rendering information vector specific settings are provided via `TSLDirectImportVectorSettings`.

Styling information for vector data is provided as a `TSLFeatureClassConfig`. This information may be set on a per data set basis and may include rendering specific to each scale band. A feature configuration may be created through the MapLink API or by exporting a MapLink Studio feature book as an MLD File.

The `TSLFeatureClassConfig` and associated classes provide many of the concepts used by MapLink Studio, including:

- A hierarchical list of features
- Different configuration for features based on product specification/detail level. When used in the Direct Import SDK product specifications must be set on the dataset prior to loading via `TSLDirectImportDataSet::product`.
- Feature masking
- Automatic feature classification, for example with either a single feature per attribute value or classification based on a range of values
- Multiple levels of feature classification
- Text label generation based on attribute values
- Data Analysis

The direct import layer provides functionality to analyse a dataset and produce an initial `TSLFeatureClassConfig`. This will populate the feature configuration with a list of features found in the data.

If present in the data, and supported by the direct import driver, the feature configuration may include feature classification, masking and rendering information.

This analysis can often take a long time as it requires iterating over all of the source data. This should be performed as an offline process to produce a feature configuration for the data or product. Alternatively, the feature configuration may be exported from the MapLink Studio feature book.

2.5.6 Raster specific settings

Any raster specific settings for a data set are provided via `TSLDirectImportRasterSettings`.

2.5.7 Caching

2.5.7.1 In-Memory Cache

The in-memory cache will store processed and displayed data in memory. Data will be prioritised based on the most recently drawn area of the world and will automatically be swapped to the on-disk cache when required. The cache size will directly affect the display of vector data, and processing of both vector and raster data. If the in-memory cache size is too small, it may trigger a high amount of disk input/output memory accessing (IO) when panning the map display.

2.5.7.2 On Disk Cache

The on-disk cache will store processed data on disk, along with the parameters used to create the data. Like the in-memory cache, data will be prioritised based on the most recently drawn area of the world. This cache may be left on disk once the data layer is destroyed and re-used in a future run of the application. Any data which is loaded with the same settings as before will be loaded from disk, instead of being processed from the source data. The cache size will affect the amount of disk space used by the layer. If the on-disk cache size is too small it will cause the data to be processed from source, which may delay the appearance of data on the display.

2.5.7.3 Raster Draw Cache

The raster draw cache is used to cache raster data when drawing. The cache size will affect the amount of raster data which can be displayed at a time. If the raster draw cache size is too small raster data may not be drawn and will greatly reduce performance of the map display.

© 2022 Envitia Ltd

2.5.8 Optimising Raster Data for Direct Import

One of the standard Direct Import Drivers for MapLink Pro uses GDAL/OGR to load the data. This allows a user to take advantage of GDAL command line utilities to optimise the data for use in the runtime environment.

2.5.8.1 Creating Overview Layers

A common way to allow an application to load raster images with high performance is to produce reduced resolution versions of data that are used when the display is at an appropriate scale. Some formats can have these overview layers inherently within for the format specification, others do not support it or leave it as optional. MapLink Studio does this automatically by default for processed maps.

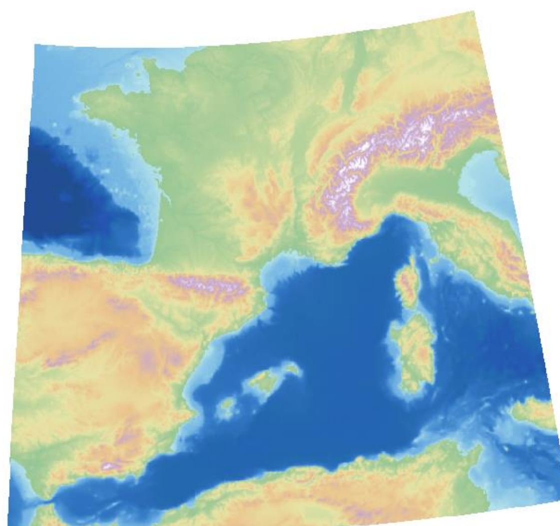
GDAL/OGR provides the '`gdaladdo`' command line utility which allows you to create overview layers which sit alongside a raster image but are automatically picked up when the raster is loaded in the Direct Import SDK. Note that GDAL does not support interpolation of 8-bit palette images, so producing overviews for this kind of data may improve performance but reduce the quality.

2.5.8.2 Combining Raster Mosaics

One common scenario is for a related set of raster images to be supplied as individual tiles. This can be cumbersome to manage in an application. GDAL/OGR has the concept of a 'Virtual Raster', which is made up of a group of rasters but behaves to the application to like a single image. The command line utility to produce this is '`gdalbuildvrt`'. The options to this utility are flexible and can also be used in tandem with other utilities. The following sequence allows a mosaic of terrain files to be loaded.

- Create a list of files that make up the mosaic. On Windows, from a folder containing subfolders with DTED .dt0 files, this might be:
    - "`dir /b /s /a-d > files.txt`"
- Combine those into a single file that can be loaded into the Direct Import Data Layer:
    - "`gdalbuildvrt -input_file_list files.txt dted.vrt`"
- Style the DTED files using a colour relief:
    - "`gdaldem -color-relief -of VRT dted.vrt <MAPLINK_HOME>\config\colourramps\elevationCombined.ctr styled_dted.vrt`"

The '`styled_dted.vrt`' should be loaded into the Direct Import Data Layer as a single styled raster, producing an image such as:



2.6 Application Architecture

MapLink has almost unparalleled flexibility among GIS components. At the heart of this lies the fact that MapLink is essentially passive. It does not create any windows, nor trap any events. The drawback of this design is that a few extra steps are required in your code. However, the benefit is that you do not have to design the rest of your application around MapLink; it will fit into whatever architecture is most suited to your particular problem domain.

2.7 The View Model

All user interface elements in an Android application are built using the Android `View` and `ViewGroup` objects. Each view contains some data that is drawn to the screen, while the `ViewGroup` acts as a container for multiple `View` objects.

In MapLink, data layers can be thought of as an equivalent to a `View` object. The Core SDK contains several specialisations of the base `TSLDataLayer`, each capable of displaying different types of data. The most common is the `TSLMapDataLayer`. This manages and displays maps that have been generated by MapLink Studio. Another common data layer is the `TSLStandardDataLayer`. This is typically used to display simple vector overlays.

The MapLink drawing surface acts as a `ViewGroup`, containing one or more data layers to visualise. Unlike a `ViewGroup` a drawing surface cannot contain other drawing surfaces. As described above, MapLink is passive so relevant events should be managed by the application and passed onto MapLink.

The data layers supported by MapLink Pro for Android are:

- Map data layer
- Standard data layer (vector data)
- Raster data layer (PNG, TIFF and Jpeg)
- Web Map Server (WMS) data layer
- Web Map Tile Server (WMTS) data layer
- Custom data layer (for application defined drawn data)
- Latitude/longitude grid data layer
- CADRG data layer
- Layers imported from an OWSContext
- Direct import data layer

© 2022 Envitia Ltd

2.8 Error Handling

In keeping with its passive nature, MapLink will not interrupt your application by displaying its own error dialogs.

Instead of interrupting program flow, MapLink maintains an internal stack of errors that have occurred and returns a failure status from those methods that fail. The error stack may be accessed through the `TSLThreadedErrorStack` utility class. Alternatively, an application can be notified of these errors as they occur using the `TSLMapLinkErrorCallbackInterface`.

Error reports on the error stack are encoded as an error number with an associated string to provide further information, such as a filename. These allow the application to identify the error that has occurred and handle it in a graceful manner.

The error numbers may be easily translated by calling `TSLThreadedErrorStack.lookupMessage` or by looking the number up in the `'msg'` files contained in the config directory. Full error information will not be available before the standard configuration files have been loaded.

2.9 View and Interaction Modes

Most applications will require some sort of navigation controls such as pan and zoom. To assist with rapid application development, Envitia supply two interaction modes designed to operate in conjunction with Android's standard `Listener` and `MotionEvent` interfaces and a MapLink drawing surface.

2.10 Coordinates and Positions

MapLink uses several different types of coordinate storage. These are optimised for particular uses.

Map units are the basic Cartesian units generated by the coordinate system applied to a map within MapLink Studio. These are dependent upon the transformations applied, but are generally mapped to nominal metres in the chosen map coordinate system. The origin of this coordinate space is typically the centre of the map projection that has been applied, modified by any false easting and northing.

Related to map units, user units are available through the Core SDK. They are simply scaled and offset versions of map units. For example, if map units were in metres a user unit scale of 1000 could be applied so that the map may be referenced in kilometres. Likewise, the origin could be offset to the centre of the map space, the lower left corner or some other user-defined position. Generally, the current view of a drawing surface is specified in terms of user units.

Latitude and longitude are typically used for real world positions. It is important to note that latitude/longitude coordinates are in themselves of no value since they are only applicable to a particular reference datum. In MapLink this is assumed by default to be the reference datum of the map's output coordinate system. A flag on the conversion functions allows this to be referenced via WGS84 instead. The drawing surface contains methods that allow the current view to be set by specifying a latitude/longitude position and a range in map units. This is particularly useful when using the dynamic projection capabilities of MapLink, since the underlying coordinate system, and hence the map unit scale and origin may change.

TMC units are the internal integer units that MapLink uses to store its geometry. These are independent of map and drawing surface and are used to create new geometric primitives.

Device units are the reference system of the output display device that a drawing surface is attached to. On Android these will typically be pixels.

The drawing surface and map data layer classes contain methods for conversion between these various coordinate systems.

© 2022 Envitia Ltd

2.11 Configuration Data

MapLink holds style information in various configuration files. These configuration files provide a mapping between internal styles or colour IDs and their visualisations.

Standard configuration files for line styles, fill styles, fonts, symbols and colours are provided in the installed MapLink `config` directory (see sections 5.1 and 5.12). When a style is attached to a feature class or to an entity it stores the style ID. The run-time rendering engine looks the ID up in the currently loaded style list.

The configuration data is held statically so need only be loaded once per application run, during application startup.

For more information about packaging and loading the MapLink configuration on Android please see section 5.12.

3 INSTALLATION

3.1 Prerequisites

It is assumed that an Android Studio based Android development environment is already setup. It is also assumed that the required version of the Android SDK has already been downloaded via the Android SDK manager tool. Please refer to the official Google documentation for information on how to do this.

3.2 Requirements

Below is the list of requirements that must be fulfilled before the use of MapLink for Android SDK:

- An Android device running *at least* the Android 4.4 (API level 19) operating system
- The device must also support OpenGL ES 2.0

Note that these requirements are an absolute minimum. We recommend using a device with at least 2 CPU cores and 1GB of RAM. If an application is loading large amounts of data (especially raster data) via the Direct Import SDK or other data layer we recommend using a more powerful device, or tailoring the application to the intended target devices.

MapLink should operate correctly in Android emulators as long as they have sufficient OpenGL ES support. As of MapLink 10.2 there are no known issues with Android emulators. Note that these emulators are often faster than physical devices so care must be taken when considering the performance of an application.

3.2.1 Supported Architectures

MapLink Pro for Android supports the following device architectures:

- Armeabi-v7a (Arm 7)
- Arm64-v8a (Arm 8)
- x86
- x86_64

If you require support for additional architectures please contact sales@envitia.com

3.3 Installation Layout

- `Config` – The MapLink config directory for use with android applications
- `Documentation`
    - `JavaDoc` – Compiled javadoc for all included SDKs, as a directory and .jar
    - `Guides` – Development guides for MapLink
    - `Licences` – Licence files for MapLink components and sample data
- `Examples`
    - `Prebuilt` – Prebuilt versions of the example applications
    - `BasicMapLinkApplication` – Basic Sample application, see section 10
    - `BasicMapLinkApplicationTracks` – A more complicated sample with APP6A symbols, see section 11
- `Java` – Java libraries for MapLink SDKs
- `Lib` – Native libraries for MapLink SDKs
- `ThirdParty` - Source code for third party components
- `Maps` - Sample MapLink Studio maps

3.4 Getting started

It is recommended that the contents of the Envitia supplied medium are copied (or in the case of a zip file, extracted) to a local hard disk, for example: *C:\MapLink_Android*.

This is all that is required to begin linking *MapLink for Android* to your Android applications. For the rest of this guide, the directory on the local hard disk shall be referred to as `$(MAPL_ANDROID_HOME)`.

4 Including MapLink Pro for Android When Building your Application

There are two components required to include MapLink functionality in an Android application:

- MapLink Java Archives: *.jar* files
- MapLink native libraries: *.so* files

4.1 Importing the MapLink CoreSDK into an Android Studio Project

All Android applications built upon MapLink for Android must include a dependency on `MapLink.jar`, which is located in the `$(MAPL_ANDROID_HOME)/Java folder.`

The CoreSDK JNI native libraries (.so files) must also be included in the Android application. These are located in the `$(MAPL_ANDROID_HOME)/lib` folder.

An example project configuration may be found in the installation under `$(MAPL_ANDROID_HOME)/examples` or in section 5.1.

4.1.1 Why Aren't the Native Libraries Inside the Jar File?

The native libraries are not packaged within the jar files to allow more flexibility when building an application. In most situations an application will not require all of the SDKs to be included, and may not require libraries for all supported architectures to be included. Any files which are not required by an application should be excluded from the APK in order to reduce installation size.

4.2 Additional MapLink SDKs

Other MapLink SDKs, such as the TerrainSDK, are structured in the same manner, with a single `.jar` file and one or more `.so` files.

These should be added to the project in the same manner as the CoreSDK. The required libraries for additional SDKs are located next to the CoreSDK in the installation.

The method call to load the native libraries will need a small modification to include additional SDKs and the licence for these must also be unlocked before use. Please see section 5.1.

Only include libraries that are required for the correct functioning of your Android application. If unnecessary libraries are included in your application the resulting package will be larger than it needs to be.

4.2.1 WMTS Data Layer

Use of the WMTS data layer does not require an additional .jar file but does require that the following .so files are included in the application:

- `libMapLinkOWS.so`
- `libttlwmts.so`
- `libMapLinkWMTSDL.so`
- `libMapLinkWMTSDLJNI.so`

These libraries will be loaded automatically when a `TSLWMTSDataLayer` object is created.

4.2.2 CADRG Data Layer

Use of the CADRG data layer does not require an additional .jar file, or additional .so files.

The CADRG data layer will require an additional component in the licence key. Please see section 5.1.

4.2.3 Terrain SDK

Use of the Terrain SDK requires the following files:

- `maplinkterrain.jar`
- `libMapLinkTerrain.so`
- `libMapLinkTerrainJNI.so`

The Terrain SDK will require an additional component in the licence key.

4.2.4 GeoPackage SDK

Use of the GeoPackage data layer requires a dependency on the `maplinkgeopackage.jar` file (located in `$(MAPL_ANDROID_HOME)/Java`) and the following JNI native libraries (located in `$(MAPL_ANDROID_HOME)/lib`):

- `libMapLinkGeoPackage.so`
- `libMapLinkGeoPackageJNI.so`

4.2.5 OWS Context SDK

Use of the OWS context data layer requires a dependency on the `maplinkowscontext.jar` file (located in `$(MAPL_ANDROID_HOME)/Java`) and the following JNI native libraries (located in `$(MAPL_ANDROID_HOME)/lib`):

- `libMapLinkOWSContext.so`
- `libMapLinkOWSContextJNI.so`

The following libraries provide the ability to load the various OWSContext offering types. Although these libraries are optional we recommend including them for full OWSContext functionality.

- `libOWCGMLOffering.so` – Required for GML offering support
- `libOWCWMSOffering.so` – Required for WMS offering support
- `libOWCWMTSOffering.so` – Required for WMTS offering support

4.2.6 Direct Import SDK

Use of the direct import data layer requires a dependency on the `maplinkdirectimport.jar` file (located in `$(MAPL_ANDROID_HOME)/Java`) and the following JNI native libraries (located in `$(MAPL_ANDROID_HOME)/SDK/lib`):

- `libMapLinkDirectImport.so`
- `libMapLinkDirectImport_gdal.so`
- `libMapLinkDirectImportJNI.so`

4.3 Importing JavaDoc for MapLink into Android Studio

To view the JavaDoc documentation for MapLink classes and methods you will need to add MapLink's `javadoc.jar` as an attachment to the `maplink.jar` library. This will

provide helpful information and context in the IDE during development. `javadoc.jar` is located in the `$(MAPL_ANDROID_HOME)/Documentation` folder and contains Java documentation for all MapLink SDKs. An HTML version of this documentation is also included in the installation.

5 MapLink Components and Concepts

Most applications that use MapLink use a set of common components from the MapLink API. This section discusses these components, and how they might be used by an application.

5.1 Things Every Application Must Do

There are a few things that every application must do in order to use MapLink:

- Load the MapLink native libraries
- Initialise the MapLink environment
- Load MapLink's configuration
- Unlock the required MapLink components using a licence key
- Some SDKs such as the OWSContext SDK require additional initialisation steps in order to load and register any plugins included with the application

Section 10 'Sample Application Walkthrough - Basic MapLink Application' describes how to implement these steps in a sample application.

5.1.1 Load the Native Libraries

Loading of MapLink's native libraries must be done before any MapLink classes can be used. This is usually done inside a static initialiser in the application's main class:

```
static {
  nativeLibrariesLoaded = TSLUtilityFunctions.loadNativeLibraries();
}
```

`TSLUtilityFunctions` is provided by the `com.envitia.maplink.core.utility` package. By default `loadNativeLibraries()` loads the libraries for the CoreSDK only. If further SDKs are required, the additional SDK(s) should be supplied using the `TSLMapLinkLibraries` enumeration. Multiple additional SDKs may be loaded at once but the application should only make one call to `LoadNativeLibraries`:

```
nativeLibrariesLoaded =
  TSLUtilityFunctions.loadNativeLibraries(
     new TSLMapLinkLibraries[]{ TSLMapLinkLibraries.TerrainSDK }
  );
```

If this method returns `false` it means that the application has not been packaged correctly according to the steps in section 4 and MapLink cannot be used. In this case an error will also be printed to `logcat` including the specific library that could not be loaded.

5.1.2 Initialise the MapLink Environment

`TSLMapLinkEnvironment` provides MapLink with information on where to store temporary files, where to create caches and provides access to the application's context in order to retrieve information from the Android environment.

Applications may accept the default settings of this class, however it is recommended that these are reviewed for every application.

`TSLMapLinkEnvironment` implements the singleton design pattern. Use the `instance()` method to get the singleton object. Before accessing the singleton object, the application must first call `TSLMapLinkEnvironment.initialise`.

MapLink is able to load and process a large amount of data. Consequently MapLink may require a large amount of temporary or cache storage. Android applications may

© 2022 Envitia Ltd

encounter problems when managing a lot of data as the Android system will clear the application's cache directories if disk space is low.

If an application is going to load large amounts of data we recommend distributing the data on external storage or setting the asset decompression directory to a location on external storage. This will ensure that any temporary or cached files are not deleted while the application is being run.

Any data stored within the application's assets directory must be cached before loading. This is because MapLink is mostly implemented natively and has no direct access to the assets directory.

In previous versions of MapLink applications could use the `TSLFileLoaderAssetManager` to load data from the assets directory. This class has been deprecated and may no longer be used when loading configuration files. The `TSLFileLoaderAssetManager` may still be used when loading data into a data layer however this is not recommended.

5.1.3 Load the Standard Configuration

Next, an application should tell MapLink to load its configuration files. This is accomplished by a call to `TSLDrawingSurfaceFactory.loadStandardConfig`. This only needs to be done once per application, during application startup. Methods of packaging MapLink's configuration files with an application are discussed in section 5.12.

5.1.4 Unlock MapLink Components

Before a MapLink component is used, it must first be unlocked with a corresponding licence key. This is done by calling `TSLUtilityFunctions.unlockSupport`. See the sample application initialisation in section 10.5.

The call to `unlockSupport` may happen before or after loading the standard configuration, but must occur before the use of any components which require a licence. A `TSLComponentNotLicenced` exception will be thrown if an un-licenced component is used.

5.2 The Drawing Surface

All visualisations performed by MapLink are performed through a drawing surface. Therefore an application that wishes to use any of MapLink's drawing capabilities needs to have one. The drawing surface classes are provided in the `com.envitia.maplink.core.drawingsurfaces` package.

Since the drawing surface forms part of the user interface for an application, MapLink provides a ready-made `View` object called `TSLEGLSurfaceView`, built on top of Android's `GLSurfaceView`. This lets an application use the normal Android XML layout mechanism to include a drawing surface in the user interface.

As the `TSLEGLSurfaceView` extends Android's `GLSurfaceView` it inherits several useful characteristics. The most important of these characteristics is that all rendering occurs in a dedicated thread, not in the application's UI thread. This allows for significantly more responsive applications as the application will not have to wait for any rendering in progress to complete in order to respond to user input. As the MapLink drawing surface resides in this thread, applications will need to take care to ensure operations that are not thread safe occur in the correct thread - this is covered in section 7.

5.3 Data Layers

As mentioned in the introduction, MapLink data layers are the component used to visualise various types of data such as a map created by MapLink Studio or some user-defined 2D geometry. A very simple viewer application might only contain one data layer. More complex applications will contain more layers of varying types.

Generally, data layers are divided into two categories: background layers and overlay layers. Background layers are usually used to provide context to data in the overlay layers. For example, a map would be used to give positional context to a set of location markers. This distinction is not strict: data layers can be used in any desired configuration in your application. The distinction exists simply to help visualise how the layers can be used.

The majority of the common data layer classes, including the base interface `TSLDataLayer`, are located in the `com.envitia.maplink.core.datalayers` package.

5.3.1 Background Layers

The `TSLMapDataLayer, TSLWMSDataLayer and TSLWMTSDataLayer` data layers are examples of background layers. These are usually the bottommost layer in the drawing surface, with other types of data layer drawn on top.

Background layers are sometimes referred to as *coordinate providing layers* - layers that have a defined coordinate system. The explanation of what a coordinate system is beyond the scope of this document - readers should refer to the MapLink Studio User Guide section 11 and 'MapLink Pro Developer's Guide' section 3.2 for an overview. These layers control the coordinate system for the drawing surface, and therefore define what Map Units are for that surface.

A drawing surface can have multiple coordinate providing layers present in the drawing surface at the same time. In this case the coordinate providing layer added to the drawing surface last defines the coordinate system for the drawing surface and the coordinate system for the other layers is ignored. When using multiple coordinate providing layers in this fashion it is necessary to ensure that the layers use both the same coordinate system *and* have the same TMC per map unit value, otherwise the non-coordinate providing layer will not appear in the correct place.

5.3.1.1 Web Map Service DataLayers

The MapLink Web Map Service (WMS) data layer allows loading of OGC standardised web map servers, refer to the 'MapLink Pro Developer's Guide' for more detail.

The MapLink Web Map Tile Service (WMTS) data layer allows loading of OGC compliant Web Map Tile Servers and functions in a similar fashion as the WMSDataLayer.

On Android, use of the `TSLWMTSDataLayer` and other classes in the `com.envitia.maplink.core.datalayers.wmts` package requires that the application package includes the following native libraries, which are included by default with the Core SDK:

- `libMapLinkOWS.so`
- `libttlwmts.so`
- `libMapLinkWMTSDL.so`

These libraries are loaded automatically when required by the WMTS data layer and can safely be excluded to reduce an applications size if no WMTS functionality is required.

5.3.2 Overlay Layers

The `TSLStandardDataLayer`, `TSLRasterDataLayer` and `TSLCustomDataLayer` are all examples of overlay layers. There are usually placed on top of a background layer and are used to display application domain-specific information.

Overlay layers are not normally coordinate providing layers (the `TSLCustomDataLayer` can be made coordinate providing by an application if required, but is not by default). This means that the positions of items specified in TMCs are tied to the current coordinate providing layer in the drawing surface. For example, a `TSLStandardDataLayer` containing items positioned by using the MapLink latitude/longitude to TMC conversion functions when a particular map was loaded into a `TSLMapDataLayer` would appear in a different place if a map using a different coordinate system was loaded into the `TSLMapDataLayer`. If this happens the application will need to reposition all of the items in the `TSLStandardDataLayer` using the new coordinate system in the drawing surface.

5.4 Java enums

The `enums` used by the MapLink SDKs natively do not all translate to sequential numbers, as such the numeric values of the Java `enums` are also not sequential.

This means that the `ordinal()` method does not return the equivalent native value.

If the numeric value of an `enum` is required, for example when calling the `setRendering(TSLRenderingAttributeInt,int)` methods, then the `getMapLinkValue` method of the `enum` should be used.

5.5 2D Vector Geometry

The MapLink geometry model maps directly onto the OpenGIS simple features model. The MapLink concept for an instantiated piece of geometry is an entity. These are accessed through classes derived from `TSLEntity`, with each different type of geometry having its own class.

Note that there is a distinction in MapLink between geometry and rendering. The geometry defines the topography of an object – where it is in the world. The rendering defines the visualisation of that object. The geometry is always an inherent part of the entity, whereas the rendering may be stored on the entity, or separately on a drawing surface or data layer. Rendering is discussed in further detail in section 5.6.

Several primitives define angles for rotation or reference points. These are measured with zero degrees as the x-axis and positive anti-clockwise.

The 2D vector geometry classes are found in the `com.envitia.maplink.core.geometry` package.

5.5.1 TSLEntity

This is the base class for all 2D geometric primitives. It gives access to the common methods of all entity types including rendering definitions, attribute interrogation and cross-entity spatial queries. It gives no access to the geometric coordinates, since these are dependent upon the derived class.

5.5.2 TSLPolyline

This is a single dimensional line, which has length, but is assumed to have no area. It is typically used to represent such real world features such as roads, rivers, railways, routes, cables and boundaries. A polyline must have at least two points, but other than that there are no limitations placed upon the coordinates.

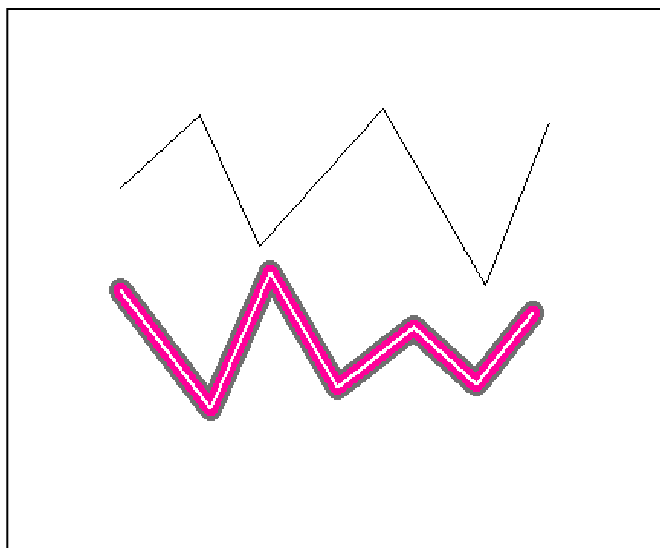


Figure 1 Polyline

5.5.3 TSLPolygon

A polygon is a two dimensional surface. It therefore has an area and a perimeter. The rendering of a polygon may actually include a hollow fill so only the edge may be visible. A polygon may have holes, which in MapLink terminology are called 'inners'. A valid polygon has some restrictions placed upon the geometry so that it conforms to OpenGIS definitions. The coordinates that define the outer or inners of a polygon must have no consecutive duplicate points, and the edges may touch but not cross. The inners must not overlap any other inner, or the outer.

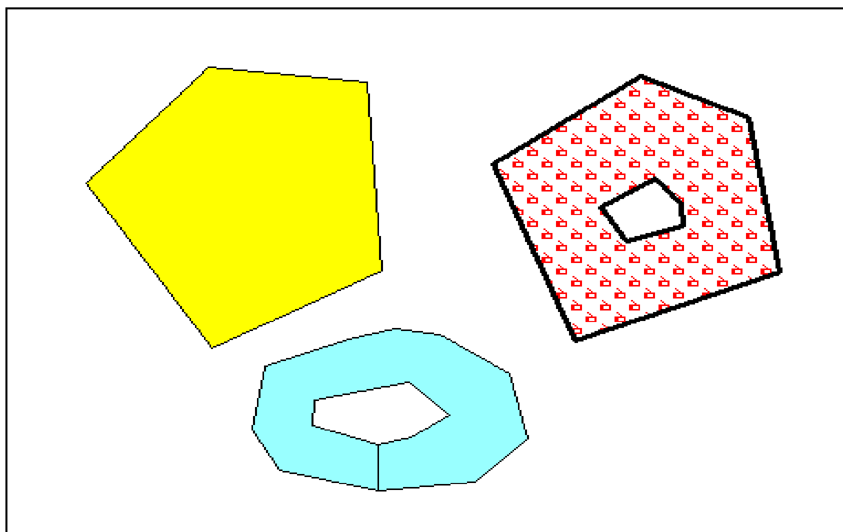


Figure 2 Polygon

MapLink has one important difference from the OpenGIS specification however. In MapLink, the coordinates of an inner or outer ring may touch along an edge, rather than at a point.

5.5.4 TSLText

The `TSLText` object consists of a single position coordinate and a text string. Each text primitive may have a horizontal or vertical alignment which dictates where the text is drawn relative to the specified position. Text may be rotated and sized. Since the font style and scaling have a large effect on the rendering of the piece of text, the extent of the text primitive is held separately for each Drawing Surface that has a unique id.

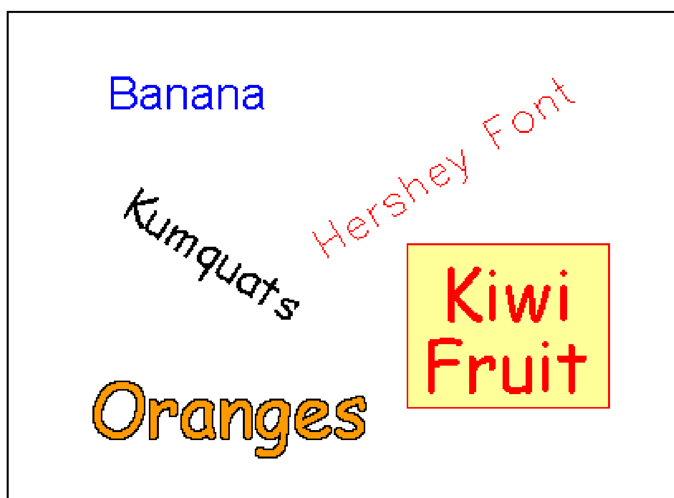


Figure 3 Text

Text primitives in maps are held in a separate sub-layer within the map and are always drawn after the polygons and polylines. This is to prevent text close to tile edges being overwritten by the polygons that exist in the adjoining tile.

© 2022 Envitia Ltd

A single text object may be split over several lines, by including a newline amongst the text. Any alignment and background will take all lines into account.

5.5.5 TSLSymbol

Like `TSLText` objects, symbols are specified geometrically by a single coordinate. The zoom level of the drawing surface and the rendering attributes attached to the entity can significantly affect the extent of a symbol. Because of this, symbols also hold their extent separately for each uniquely identified drawing surface.

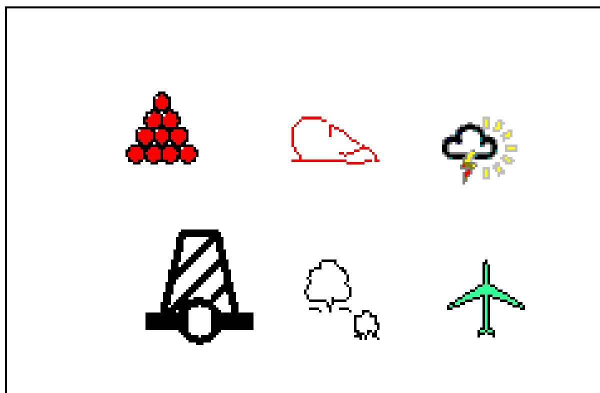


Figure 4 Symbols

There are two different types of symbols available in MapLink - vector and raster.

Vector symbols are scalable and are held in individual TMF files – a proprietary MapLink format. Vector symbols have the ability to display text, which may be dynamic (see the section following).

Raster symbols are held in supported image formats (e.g. PNG), with one symbol per file.

Symbol primitives in maps are held in a separate sub-layer within the map and are always drawn after the polygons and polylines. This is to prevent symbols close to tile edges being overwritten by the polygons that exist in the adjoining tile.

5.5.6 TSLEllipse

A `TSLEllipse` is a two-dimensional surface that has area and perimeter. It is defined geometrically by the centre point, x and y radial distances and rotation angle. The radial distances are those before rotation is applied. MapLink currently has no facilities for partial ellipses such as chords or sectors. `TSLEllipse` objects typically do not appear in map data and are unlikely to be produced by MapLink Studio.

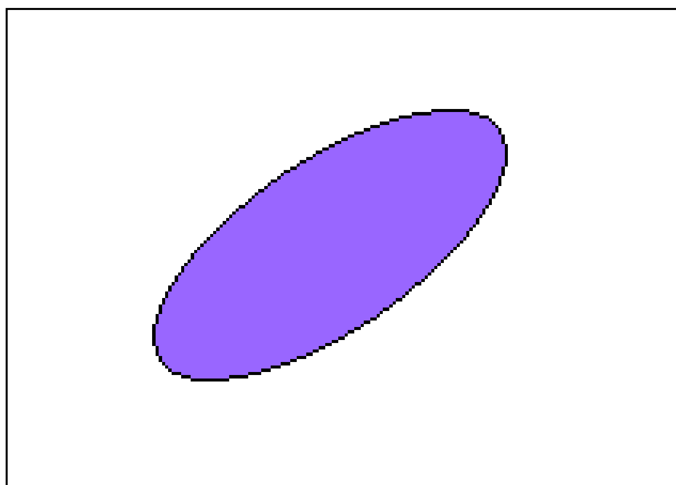


Figure 5 Ellipse

© 2022 Envitia Ltd

5.5.7 TSLArc

The TSLArc primitive is a one-dimensional curve, which is a portion of the circumference of an ellipse. It therefore has length but no area. It is specified geometrically by the centre of the ellipse, the x and y radial distances and the start and end angle of the sweep. The radial distances and angles are those before rotation is applied. An additional rotation attribute allows the source ellipse to be rotated. The sweep of the arc is anti-clockwise from start angle to end angle. TSLArc objects typically do not appear in map data and are unlikely to be produced by MapLink Studio.

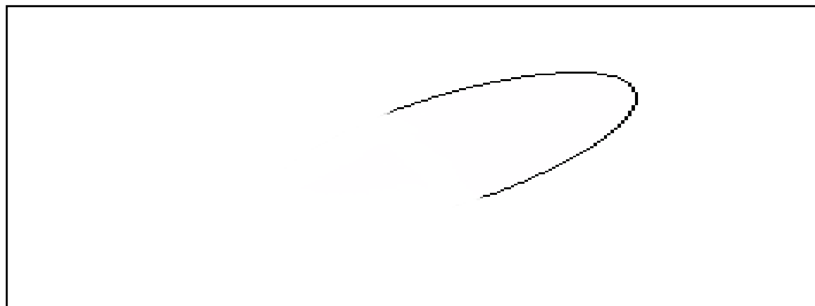


Figure 6 Arc

5.5.8 TSLRectangle

This type of geometric primitive is specified by two corners and a rotation angle. The TSLRectangle may be rotated about its centre.

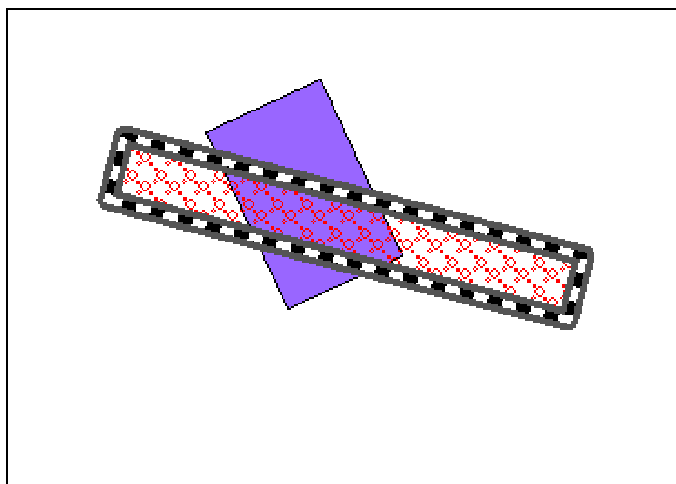


Figure 7 Rectangle

5.5.9 TSLEntitySet and other Collections

This is a collection of other entities. Note that an entity set can contain other entity sets and thus be hierarchical. It has no geometric attributes of its own but inherits its envelope as the union of its children's envelopes.

Unlike OpenGIS collections, a TSLEntitySet can contain different types of TSLEntity.

Simple single-type collections are available via the TSLMultiPolygon, TSLMultiPolyline and TSLMultiPoint classes. These represent a single entity and as such the constituent parts only have limited access to the geometry and are not derived from the TSLEntity class.

5.6 Rendering Configuration

Rendering is a term used for the graphical properties used to define the visual appearance of an entity. MapLink has very powerful and flexible facilities for visualisation. Rendering may be defined in three different places: on individual entities, on data layers or on drawing surfaces. The first method is commonly called 'entity based rendering' whilst the other methods are 'feature based rendering'.

5.6.1 Entity Based Rendering

Each entity within MapLink may have its own unique rendering defined. This takes precedence over any feature based rendering that may have been configured and is typically used for overlays in a `TSLStandardDataLayer`.

Entity based rendering is configured using the `TSLEntity.setRendering` method and the `com.envitia.maplink.core.rendering.TSLRenderingAttributes` class, which specifies all of the new rendering attributes to set. The current rendering attributes may be queried with the `TSLEntity.getRendering` method.

A summary of the available rendering attributes is provided in section 5.7.

5.6.2 Feature Based Rendering

Data layers often contain lots of entities that need to be rendered in a similar fashion. Feature based rendering allows the rendering styles to be defined only once for a particular map feature type and then specific entities to be tagged with an identifier to indicate what feature type it represents.

As an example of feature based rendering, MapLink may be told that features of type “A Road” are to be drawn as red lines with black edges, and individual entities are tagged as being an “A Road”. In a map, the rendering is usually configured within MapLink Studio, using the feature book. In a run-time application, it may be configured on the on the `TSLDataLayer`. Wherever feature based rendering is configured, it uses the same `setFeatureRendering` method.

This method takes the feature name, feature ID and a `TSLRenderingAttributes` object, which specifies the new attributes to set. Note that the feature name is optional. If null is passed the feature ID is used. Methods which set an individual attribute are also present and use the `TSLRenderingAttributeXXX` enums.

5.6.3 Determining the Source of Rendering Attributes

As described above, there are multiple places to define the rendering attributes of an entity. MapLink must determine where to fetch the attributes from at run-time.

When rendering an entity, MapLink first of looks to see if there is any entity based rendering defined on the entity. If so then that is used. If none exists then the feature ID stored on the entity is used to search for feature based rendering on the `TSLDrawingSurface` currently being drawn. If none exists on the `TSLDrawingSurface` then the `TSLDataLayer` is searched. If there is also no feature based rendering defined there then the process begins again starting at the parent of the entity - the `TSLEntitySet` that contains it.

If MapLink cannot determine the rendering attributes then the entity is not drawn. All rendering attributes for an entity will be taken from the same place. For example, it is not possible to define the edge colour of a polyline using entity based rendering and the edge style using feature based rendering.

5.7 MapLink Rendering Attributes

Wherever they are defined MapLink's graphical properties are split into five categories and three types. These are used to control the appearance of all MapLink entities, regardless of their source.

On Android, colours are specified as 24-bit rgb values, as returned from `android.graphics.colour.rgb`. Colour IDs from `tslcolours.dat` may be used, but must be converted to an rgb value first using `TSLDrawingSurfaceBase.getColourValue`.

The ‘not set’ value for colours when using the Java API is 0, not -1 as it is natively. A colour set to -1 will display as white.

5.7.1 Generic Attributes

These are available on all Entities, regardless of type. They are

- `TSLRenderingAttributeInt.FeatureID`: Signed 32-bit value, user defined features may be from 1 to 16777215 (0xFFFFFF). This value is used to lookup feature based rendering that may be applied to an entity. The default is 0.
- `TSLRenderingAttributeInt.RenderLevel`: Valid values are -5 to +5. The default is 0.
- `TSLRenderingAttributeBoolean.Visible`: Boolean flag which indicates whether the entity should be drawn. The default is true.
- `TSLRenderingAttributeBoolean.Selectable`: Boolean flag that indicates whether the entity can be found when searching the data using the find and query methods of the drawing surface and data layer. Note that the data layer properties `TSLProperty.Detect` and `TSLProperty.Select` are also taken into account when searching and selecting. The default is true.

5.7.2 Line Rendering Attributes

These are available on one-dimensional entities such as polylines and arcs. They are:

- `TSLRenderingAttributeInt.EdgeColour`: This value must be a integer RGB value returned from android.graphics.color.rgb(). The default is 0, which inhibits display of the entity.
- `TSLRenderingAttributeInt.EdgeStyle`: This value must be an index from the tsllinestyles.dat file. The default is –1, which inhibits display of the entity.
- `TSLRenderingAttributeInt.EdgeThicknessUnits`: This value must be one of the `TSLDimensionUnits` enum values. Use of this attribute allows the line thickness to be defined in device units, internal TMC units, map units or points (1/72 of an inch). The default is `TSLDimensionUnits.Pixels`.
- `TSLRenderingAttributeDouble.EdgeThickness`: This value is in the units defined by the `TSLRenderingAttributeInt.EdgeThicknessUnits` value. It is a floating point number so when applicable may hold fractional values. Note that complex and multi-pass line styles have a minimum device unit thickness in order to maintain a coherent display. If an attempt is made to set a smaller thickness, or a variable thickness line produces a smaller value, then the minimum is used. The default is –1, which inhibits display of the entity.
- `TSLRenderingAttributeInt.EdgeOpacity`: This value controls the transparency of the line. Values range from 0 to 32767 where 0 is fully transparent and 32767 is fully opaque.

© 2022 Envitia Ltd

5.7.3 Area Rendering Attributes

These are available on two-dimensional entities such as polygons, ellipses and rectangles. The rendering for the edges of areas are different from those used for lines – this is because there may be both lines and area features assigned the same feature code. The current area rendering attributes are:

- `TSLRenderingAttributeInt.FillColour`: This value must be a integer RGB value returned from android.graphics.color.rgb(). The default is 0, which inhibits display of the entity.
- `TSLRenderingAttributeInt.FillStyle`: This value must be an index from the tslfillstyles.dat file. The default is −1, which inhibits display of the fill potentially leaving just the edge of the entity.
- `TSLRenderingAttributeInt.FillOpacity`: This value controls the transparency of the fill. Values range from 0 to 32767 where 0 is fully transparent and 32767 is fully opaque.
- `TSLRenderingAttributeInt.ExteriorEdgeColour` This value must be a integer RGB value returned from android.graphics.color.rgb(). The default is 0, which inhibits display of the entity. Note that this also applies to the edges of any holes in a polygon.
- `TSLRenderingAttributeInt.ExteriorEdgeOpacity`: This value controls the transparency of the edge. Values range from 0 to 32767 where 0 is fully transparent and 32767 is fully opaque.
- `TSLRenderingAttributeInt.ExteriorEdgeStyle`: This value must be an index from the tslinestyles.dat file. Note that this also applies to the edges of any holes in a polygon. The default is −1, which inhibits display of the edge potentially leaving just the fill of the entity.
- `TSLRenderingAttributeInt.ExteriorEdgeThicknessUnits`: This value must be one of the `TSLDimensionUnits` enum values. Use of this attribute allows the edge thickness to be defined in device units, internal TMC units, map units or points (1/72 of an inch). Note that this also applies to the edges of any holes in a polygon. The default is `TSLDimensionUnitsPixels`.
- `TSLRenderingAttributeDouble.ExteriorEdgeThickness`: This value is in the units defined by the `TSLRenderingAttributeExteriorEdgeThicknessUnits` value. It is a floating point number so where relevant may hold fractional values. Note that complex line styles have a minimum device unit thickness in order to maintain a coherent display. If an attempt is made to set a smaller thickness, or a variable thickness line produces a smaller value, then the minimum is used. Note that this also applies to the edges of any holes in a polygon. The default is −1, which inhibits display of the edge potentially leaving just the fill of the entity.

5.7.4 Text Rendering Attributes

These are available on Text Entities. They are:

- `TSLRenderingAttributeInt.TextColour`: This value must be a integer RGB value returned from android.graphics.color.rgb(). The default is 0, which inhibits display of the entity.
- `TSLRenderingAttributeInt.TextOpacity`: This value controls the transparency of the text. Values range from 0 to 32767 where 0 is fully transparent and 32767 is fully opaque.
- `TSLRenderingAttributeInt.TextFont`: This value must be an index from the tslfonts.dat file. The default is −1, which inhibits display of the text. Note that the

© 2022 Envitia Ltd

contents of the tslfonts.dat file are operating system dependant and so may not give an exact match if displayed on different machines.

- `TSLRenderingAttributeInt.TextSizeFactor`: This value defines the size or height of the text. It may also be adjusted by the height defined on the `TSLText` object itself. This is a floating point number, whose units are defined by `TSLRenderingAttributeInt.TextSizeFactorUnits`. The default is 0, which inhibits display of the text.

- `TSLRenderingAttributeInt.TextSizeFactorUnits`: This value is one of `TSLDimensionUnits` enum, and determines how the `TSLRenderingAttributeDouble.TextSizeFactor` value is interpreted. Typical values allow the height of the text to be defined in points, map units, internal TMC units or device units.

- `TSLRenderingAttributeInt.TextMinPixelHeight`: This value defines the minimum height, in pixels, that the text will be displayed at. It may be used for clamping text height within certain boundaries to maintain visibility. If a simple fixed pixel size is required, then use size factor units of pixels and set the size factor to be the required pixel height. The default value is 1. Note that the text may be made invisible before this value is reached, using the `TSLDrawingSurface.setDataLayerProps` method and the `TSLProperty.MinTextHeight` property.

- `TSLRenderingAttributeInt.TextMaxPixelHeight`: This value defines the maximum height, in pixels, that the text will be displayed at. It may be used for clamping text height within certain boundaries to maintain visibility. If a simple fixed pixel size is required, then use size factor units of pixels and set the size factor to be the required pixel height. The default value is 2000 pixels. Note that the text may be made invisible before this value is reached, using the `TSLDrawingSurface::setDataLayerProps` method and the `TSLProperty.MaxTextHeight` property.

- `TSLRenderingAttributeDouble.TextOffsetX`: This is the horizontal offset of the text, relative to its defined position, in addition to the alignment. This is typically used for positioning of text that has been generated relative to a point object in a map. The default value is 0. The value is interpreted according to the value of the `TSLRenderingAttributeInt.TextOffsetUnits` property.

- `TSLRenderingAttributeDouble.TextOffsetY`: This is the vertical offset of the text, relative to its defined position, in addition to the alignment. This is typically used for positioning of text that has been generated relative to a point object in a map. The default value is 0. The value is interpreted according to the value of the `TSLRenderingAttributeInt.TextOffsetUnits` property.

- `TSLRenderingAttributeInt.TextOffsetUnits`: This value is one of `TSLDimensionUnits` enum, and determines how the `TSLRenderingAttributeDouble.TextOffsetX/Y` values are interpreted. Typical values allow the offset of the text to be defined in map units, internal TMC units or device units. To keep positioning constant relative to any underlying map or associated symbol, this is usually the same as the SizeFactorUnits. The default is `TSLDimensionUnits.Undefined`, which in this case is interpreted as pixels.

- `TSLRenderingAttributeInt.TextVerticalAlignment`: Value is one of `TSLVerticalAlignment` enum. This value is only used if no alignment is stored on the entity. This is because some map data sources, such as Ordnance Survey NTF, include topographic text with defined alignments and rotations. For this rendering attribute to have any effect, the alignment stored on the entity must be `TSLVerticalAlignment.Undefined`.

- `TSLRenderingAttributeInt.TextHorizontalAlignment`: Value is one of `TSLHorizontalAlignment` enum. This value is only used if no alignment is stored

on the entity. This is because some map data sources, such as Ordnance Survey NTF, include topographic text with defined alignments and rotations. For this rendering attribute to have any effect, the alignment stored on the entity must be `TSLHorizontalAlignment.Undefined`.

`TSLRenderingAttributeInt.TextBackgroundMode`: Value is one of `TSLTextBackgroundMode` enum. This attribute allows text to be rendered with some form of background. Currently this may be in the form of a dynamically resizing rectangle, or a single pixel outline or halo around the text.

The rectangle fill colour, fill style and edge colour may be configured using other rendering attributes, but will always have a solid edge. The rectangle will dynamically resize to fit around the text and will automatically compensate for multiple lines, alignment and text size changes and will rotate will the text.

The default value is `TSLTextBackground.ModeNone`.

- `TSLRenderingAttributeInt.TextBackgroundColour`: This value must be a integer RGB value returned from android.graphics.color.rgb(). The default is 0, which inhibits display of the background. When using rectangle backgrounds, this attribute defines the fill colour. When using halo backgrounds, this attribute defines the outline colour.
- `TSLRenderingAttributeInt.TextBackgroundStyle`: Value is index from tslfillstyles.dat file. This attribute is ignored for halo backgrounds, but defines the fill style for rectangle backgrounds. The default is –1, which inhibits display of the background fill.
- `TSLRenderingAttributeInt.TextBackgroundEdgeColour`: This value must be a integer RGB value returned from android.graphics.color.rgb(). The default is 0, which inhibits display of the background rectangle edge.
- `TSLRenderingAttributeBoolean.TextRotatable`: This boolean flag enables or disables rotation of text. If the flag is false, then the rotation of the text entity and the drawing surface are both ignored when rendering the text. This is often used to inhibit rotation that has been added to map text due to coordinate system transformations. The default value is true.

Many of these attributes are interdependent.

5.7.5 Symbol Rendering Attributes

These are available on symbol entities. They are:

- `TSLRenderingAttributeInt.SymbolColour`: This value must be a integer RGB value returned from android.graphics.color.rgb(). The default is 0, which inhibits display of the entity.
- `TSLRenderingAttributeInt.SymbolOpacity`: This value controls the transparency of the symbol. Values range from 0 to 32767 where 0 is fully transparent and 32767 is fully opaque.
- `TSLRenderingAttributeInt.SymbolStyle`: This value must be an index from the tslsymbols.dat file. The default is –1, which inhibits display of the symbol.
- `TSLRenderingAttributeDouble.SymbolSizeFactor`: This value defines the size or height of the symbol. It may also be adjusted by the height defined on the `TSLSymbol` object itself. This is a floating point number, whose units are defined by `TSLRenderingAttributeInt.SymbolSizeFactorUnits`. The default is 0, which inhibits display of the symbol.
- `TSLRenderingAttributeInt.SymbolSizeFactorUnits`: This value is one of `TSLDimensionUnits` enum, and determines how the `TSLRenderingAttributeDouble.SymbolSizeFactor` value is interpreted. Typical

© 2022 Envitia Ltd

values allow the height of the symbol to be defined in points, map units, internal TMC units or device units.

- `TSLRenderingAttributeInt.SymbolMinPixelHeight`: This value defines the minimum height, in pixels, that the Symbol will be displayed at. It may be used for clamping Symbol height within certain boundaries to maintain visibility. If a simple fixed pixel size is required, then use Size Factor Units of Pixels and set the Size Factor to be the required pixel height. The default value is 1.

- `TSLRenderingAttributeInt.SymbolMaxPixelHeight`: This value defines the maximum height, in pixels, that the Symbol will be displayed at. It may be used for clamping Symbol height within certain boundaries to maintain visibility. If a simple fixed pixel size is required, then use Size Factor Units of Pixels and set the Size Factor to be the required pixel height. The default value is 2000 pixels.

- `TSLRenderingAttributeInt.SymbolRotatable`: Value is one of `TSLSymbolRotation` enum. This is more than a simple boolean flag, in order to maintain backwards compatibility. The `tslsymbols.dat` file contains a flag for each symbol indicating whether by default it should be rotatable. For example, a lighthouse symbol should remain vertical, whereas a flow arrow must be rotated to indicate the direction of flow. If your application is using the symbols in an unusual way – for example using a (non-rotatable) "airport" symbol to represent a moving "aircraft" track, then you may wish to override the standard settings.

  The `TSLSymbolRotation` enum allows you to specify that the symbol will be rotatable, not rotatable, or that the default rotatability defined in the tslsymbols.dat file should be used.

- `TSLRenderingAttributeRasterInt.SymbolScalable`: Value is one of `TSLRasterSymbolScalability` enum. This is more than a simple boolean flag, in order to maintain backwards compatibility. By default, raster symbols are not scalable and are displayed at their relevant pixel size regardless of the calculated height of the symbol. This rendering attribute allows an application to enable scaling for this raster symbol.

- `TSLRenderingAttributeLong.SymbolFontCharacter`: MapLink supports the capability to allow symbols to be characters from a font. The font is referenced via an entry in the `tslsymbols.dat` file. For such symbol styles, this rendering attribute defines the UTF-32 codepoint from the font to be displayed.

5.7.6 Raster Icon Symbols

Some of the symbols defined in the default `tslsymbols.dat` file are raster icons. These are standard image files. These have certain limitations which you should be aware of before using them:

- They are usually drawn fixed size. Regardless of the Symbol size rendering attributes, they will always be drawn as they are defined. This behaviour can be overridden using the `TSLRenderingAttributeInt.RasterSymbolScalable` attribute.

- The extent of the symbol will include the full size of the icon, not just the non-masked areas.

5.7.7 Minimum Attribute Requirements

Many of the default values inhibit display of the entity until explicitly set by the user. To enable display of the various entity types, the following rendering attributes must be set – either through entity based rendering or through feature based rendering:

- `TSLPolyline` and `TSLArc`: Requires style, colour and thickness to be set. By default the thickness is in pixels.

- TSLPolygon, TSLEllipse and TSLRectangle: A visible fill requires style and colour to be set. A visible edge requires style, colour and thickness to be set. By default the thickness is in pixels.
- TSLText: Requires a height stored on the entity of > 0, a font, a colour and a size factor. The default size factor units will multiply the entity height by the size factor to determine the TMC height of the text.
- TSLSymbol: Requires a style, colour and size factor. The default size factor units will multiply the entity height by the size factor to determine the TMC height of the symbol.

5.7.8 Why Can't I See My Object?

One of the most frustrating things that can happen when developing an application is when you expect something to happen, but it doesn't. A typical example of this in a MapLink application is an entity not appearing when it is created. There can be many reasons for the non-appearance and it can be difficult to track down. Here is a list of the most common reasons:

- The entity was never actually created. This can occur if invalid arguments are passed to the create method call – such as an empty string being passed to createText or a self-intersecting coordinate set being passed to createPolygon. Check the return value from the create call and look at the contents of the error stack to see what may have gone wrong.
- The entity has no rendering attributes associated with it. These can either be configured on the entity itself, or on the data layer or drawing surface via feature based rendering.
- The entity has insufficient rendering attributes associated with it. Even though an entity may have some attributes, they may not be enough to create a valid rendition. See Section 5.7.7 for a list of the minimum set of rendering attributes for each primitive type.
- The associated rendering attributes are illegal. This means that an index is not found in the associated configuration file. For example, using a line style index that does not exist in tsllinestyles.dat. Check the contents of the MapLink configuration files.
- Would the rendering attributes give a visible representation anyway? Some of the line styles and fill styles give no rendition – such as hollow, highly translucent or very sparse bitmap fill styles.
- Is the entity in a TSLEntitySet that is associated with a data layer? Free-floating entities are never displayed. They need to be inserted into a TSLStandardDataLayer. Is the data layer associated with the drawing surface?
- Has notifyChanged been called on the data layer after the entity is created? Without this, the data layer does not invalidate any associated buffer and so the old contents are used when drawing an unchanged view extent.
- Is the entity, its parent entity sets and associated data layer all visible? An entity can be hidden using TSLRenderingAttributeBoolean.Visible and a data layer can be hidden using TSLProperty.Visible.
- Have the entity or data layer been decluttered? An entity can be decluttered and thereby hidden, using the setDeclutterStatus method of the drawing surface. A data layer can be hidden according to zoom level using the TSLProperty.MinZoomDisplay and TSLProperty.MaxZoomDisplay properties.
- Is the drawing surface actually viewing the area containing the entity?

For text primitives, have they been hidden because they are too small or too big? These limits default to 3 pixels and 200 pixels. They can be configured using `TSLProperty.MinTextHeight` and `TSLProperty.MaxTextHeight`.

5.8 Decluttering

Decluttering is the temporary hiding of features. The features still exist in the map or data layer, but are not drawn. Applications often use decluttering, under user control, to help prevent information overload. It is applied on the `TSLDrawingSurface`.

5.8.1 Declutter Feature Name and ID

The declutter status is configured on a per-feature basis using the feature's name. The feature names are hierarchical, usually as defined using the feature subclassing configuration in MapLink Studio. Each level of the hierarchy is separated by a full-stop in the feature name. For example, a map may contain the following features:

```
vpf.Country.Europe.France
vpf.Country.Europe.Germany
vpf.Country.Africa.Egypt
vpf.Country.Asia.China
vpf.Country.Asia.Japan
vpf.Water.Sea
vpf.Water.Rivers
```

Each feature also has an associated numeric feature ID. It is this numeric ID that is stored on an entity, rather than the full name. Only the leaf features of the hierarchy have a numeric ID, others do not. For example, in the above list, "`vpf.Country.Europe.France`" has a numeric feature ID, whereas "`vpf.Country.Europe`" does not.

Decluttering may be applied at any level in the hierarchy by specifying the appropriate name. In the above example, all European countries may be decluttered by specifying "`vpf.Country.Europe`", all water features with "`vpf.Water`" and China specifically using "`vpf.Country.Asia.China`". It is for this reason that the declutter methods use the feature name rather than the feature ID.

The feature name to feature ID mapping is as defined in the feature book of MapLink Studio, or as defined on a `TSLStandardDataLayer` using the `addFeatureRendering` method. Where entity based rendering is used in a `TSLStandardDataLayer`, then the `addFeature` method may be used to provide the mapping without setting up any feature based rendering.

You can determine what features are available on a particular data layer using the `TSLDataLayer.featureList` method. This returns a read-only instance of type `TSLFeatureClassList`. This class allows the application to query the number of features available, their names and IDs. The contents of the list are typically displayed in a tree view with associated check boxes to control the declutter status.

On a `TSLMapDataLayer`, the feature class list is automatically populated when a map is loaded. On a `TSLStandardDataLayer`, it is populated by the application calling the `addFeature` or `addFeatureRendering` methods.

5.8.2 Declutter Status

MapLink uses the numeric feature ID of an entity to look up the required status before rendering the entity. The status may be set to `TSLDeclutterStatusOn`, `TSLDeclutterStatus.Off` or `TSLDeclutterStatus.Auto`. To set the status use:

```
m_drawingSurface.setDeclutterStatus( "feature", status, layer )
```

© 2022 Envitia Ltd

The layer argument is optional. If specified, it targets the decluttering at a specified data layer. This is sometimes necessary since a drawing surface may contain multiple maps generated through different MapLink Studio feature books. This means that there is a possibility of numeric feature IDs clashing. It has no direct effect on rendering since that is usually defined on the individual data layers.

The current declutter status may be queried using the `TSLDrawingSurface.getDeclutterStatus` method. This returns one of the `TSLDeclutterStatusResult` enumerations. In addition to values which map to those used to set the status, this can also return a value of `TSLDeclutterStatusResult.Partial`, when called using a non-leaf node of the hierarchy. This indicates that some Subclasses have a different declutter status.

The `TSLDrawingSurface.declutterVisible` method allows the application to query whether a particular feature is currently visible, or would be visible at a specified zoom factor. The zoom factor is specified in terms of number of user units per device unit.

5.8.3 Automatic Decluttering on Zoom

In addition to the standard 'On' and 'Off' declutter status values, it is possible to set the status to be `TSLDeclutterStatus.Auto`. This uses an additional method call to configure minimum and maximum zoom factors for which the feature will be visible. These factors are in terms of number of user units per device unit. If the current zoom factor is within the specified range then the Feature will be visible, otherwise it is invisible.

5.8.4 Declutter of Raster Features in Maps

When raster images are loaded into MapLink Studio, they may be assigned a feature name in the raster configuration panel. This feature name is then available in the usual declutter methods to enable or disable the display of that raster image. These appear in a hidden feature book section called 'Rasters' and default to 'Raster' if not supplied.

Thus to declutter all rasters in a map, with the default feature name, use the call:

```
ds.setDeclutterStatus("Rasters",TSLDeclutterStatus.Off);
```

5.9 Searching Your Data

There are several ways of searching and querying your data through the MapLink SDK – the most appropriate one depends upon what information you require and how complex your criterion for selection is.

5.9.1 Finding the Entity at a Point on the Screen

This is perhaps the most common reason for searching the data, and MapLink has a simple way of doing so using the `TSLDrawingSurface.findSelectedEntityDU` method. This takes a device unit position, such as the current cursor location, a search depth in the entity hierarchy and an aperture in device units. It returns the top-most entity found. A related method takes a position in user units.

Note: The `findSelectedEntity` methods are not currently supported for `TSLCustomDataLayers`. Please use the pick methods as documented in section 5.9.3.

A few rules are applied to the selection to make sure that the entity found is appropriate.

- An optional flag allows map data layers to be ignored. This is useful for only searching overlay layers.
- Any data layers with the `TSLProperty.Detect` property set to false and any entity with the `TSLRenderingAttributeBoolean.Selectable` attribute set to false will be ignored in the search. Note that the default value for `TSLProperty.Detect` is false.
- When searching a map data layer the currently displayed detail layer will be used for the query.

- Data layers and entity sets are searched in reverse rendering order – i.e. top-most first.
- The search will only descend the entity set hierarchy as far as the specified depth. A depth of 0 will always return the top-most entity set. A depth of -1 is a special case that will traverse all entities.
- Only entities that are visible and not decluttered will be considered.
- The distance from the specified point to the entity must be less than or equal to the specified aperture.
- For entities with complex outlines but a single `TSLCoord` position, i.e. text and symbol objects, the distance is considered to be 0 if the specified point lies anywhere within the current envelope of the entity. Note that the extents may be bigger than they appear due to font sizing with text and hidden boundaries in symbols.
- For surfaces (polygons, rectangles and ellipses) the distance is considered to be 0 if the specified point lies anywhere within the boundary of the entity (not including holes).
- If the point lies within a surface entity, and another non-surface entity has already been found to be within the aperture, then the non-surface entity will be returned in preference to the surface entity. Without this rule, it would be virtually impossible to select a polyline that is on top of a polygon since the distance to the polygon would always be 0.

5.9.2 Finding All Entities Within an Area

This is another common requirement, for which there are two pairs of query methods. One pair is on the drawing surface and the other pair is on the data layer. All of the methods allow an extent (in TMC Units) and query depth to be specified. Additionally the drawing surface methods take the name of a data layer to search.

The first method in each pair takes an optional feature name. If this is specified then only those features are considered.

The second method in each pair takes an instance of a user defined selector object – derived from the `TSLSelector` class. The selector object is called for every entity that is considered and allows user control over exactly which entities are chosen.

Where a map data layer is queried through the data layer methods, the specified extent is used to determine which detail layer is searched. An optional drawing surface ID may be used to identify which entity last rendered extent to use. Where a map data layer is queried through the drawing surface methods, the currently displayed detail layer is searched.

A few rules are applied to the selection to make sure that the entities found are appropriate.

- Any entity with the `TSLRenderingAttributeBoolean.Selectable` attribute set to false will be ignored in the search.
- Entity sets are searched in reverse rendering order; i.e. top-most first.
- The search will only descend the entity set hierarchy as far as the specified depth. A depth of 0 will always return the top-most entity set. A depth of -1 is a special case that will traverse all entities.
- Decluttering status is ignored.
- If a feature name is specified, then only those features will be considered.
- An entity is considered if its last rendered envelope overlaps the specified extent.

- If a TSLSelector object is specified, then any entity considered will be passed to the virtual select method of the object. This method should return a TSLSelectorActionType value to indicate what action to take.
- If a selector action of TSLSelectorAction.SelectNext is returned for a TSLEntitySet object, then the search algorithm will not search within the TSLEntitySet.

The query methods return an instance of the TSLMapQuery class, or null if no entities are selected. This object holds references to the chosen entities. The application can iterate through the references in the TSLMapQuery object to further act upon the entities.

5.9.3 Finding data within a TSLCustomDataLayer

Searching for data in a custom data layer can be performed using the TSLDrawingSurfaceBase.pick() methods. These query the relevant data from an applications TSLClientCustomDataLayer implementation, and optionally use a TSLSelector for further results filtering.

The following need to be implemented for this to use picking with a TSLCustomDataLayer.

- A class that inherits from TSLClientCustomPickResult. This is used as a container for whatever data the layer returns from its pick() method:

```
public class MyClientCustomPickResult implements TSLClientCustomPickResult {
  public TSLEntity entity;
  public MyClientCustomPickResult(TSLEntity entity) {
    this.entity = entity;
  }
}
```

- The pick() method on the application's TSLClientCustomDataLayer. This should populate the TSLPickResultsSet with all the data within the supplied extent:

```
public boolean pick(final TSLEnvelope extent, TSLPickResultSet results) {
  for( each entity within extent ) {
    results.addResult(new MyClientCustomPickResult( entity ));
  }
  return true;
}
```

- (Optional) A class that inherits from TSLSelector: As with the map queries, this is used to select the relevant data from the TSLPickResultSet:

```
TSLPickSelector selector = new TSLPickSelector() {
  @Override
  public TSLSelectorActionType select(TSLPickResult result) {
    if( result.queryType() == TSLPickResultType.Custom ) {
      TSLPickResultCustom customResult = (TSLPickResultCustom)result;
      MyClientCustomPickResult clientCustomResult =
        (MyClientCustomPickResult)customResult.getClientCustomPickResult();

      TSLEntity ent = clientCustomResult.entity;
      // Check that the entity intersects the required location
      TSLEnvelope boundingBox = new TSLEnvelope(200, 200, 210, 210);

      if( ent.intersects(boundingBox) ) {
        return TSLSelectorActionType.SelectExit;
      }
      else {
        return TSLSelectorActionType.IgnoreNext;
```

```
        }
      }
      return TSLSelectorActionType.IgnoreNext;
    }
  };
```

5.10 Terrain SDK

In order to use the MapLink TerrainSDK in an application, these steps must occur:

- The Java, and native TerrainSDK libraries must be included in the application, see section 4.
- `TSLUtilityFunctions.unlockSupport` must be called with a valid key, in order to unlock the Core and Terrain SDKs.

Terrain data may then be accessed by constructing the relevant terrain database class. If the terrain SDK has not been unlocked a `TSLComponentNotLicencedException` will be thrown.

```
TSLTerrainDatabase terrainDB = new TSLTerrainDatabase();
```

Information on the concepts used by the Terrain SDK can be found in the 'MapLink Pro Developer's Guide'.

5.10.1 Queries

Data may be queried from a terrain database for a single point, line or an area. The accuracy of this data depends on various factors.

Each of the query functions takes a '`highestRes`' parameter. If this parameter is true, the database will return the highest resolution data available. Note that a query for the highest detail data over a large area will take a long time.

If the '`highestRes`' parameter is false, the database will select data for the current level of detail. In order for this to be an appropriate detail level for the current view, the application must tell the database any time the displayed extent changes. For an android application this will happen whenever the view is zoomed in/out, and whenever the screen orientation changes. If the application is using the `TSLEGLSurfaceView` widget, then the draw callbacks may be used to ensure the database is kept up to date with the displayed extent.

```
@Override
  public void onPreDrawFrame(GL10 arg0, TSLEGLSurface arg1) {
    TSLEnvelope currentExtent = new TSLEnvelope();
    drawingSurfaceView.drawingSurface().getTMCExtent(currentExtent);
    if( !currentExtent.equals(displayedExtent)) {
      displayedExtent.copy(currentExtent);
      /*
       * Update the displayed extent of the terrain database. This assumes
       * that the database is in lat/lon. If the database has the same
       * coordinate system as the displayed map, then map units should be
       * used.
       */
      DisplayMetrics dimensions = new DisplayMetrics();
      getWindowManager().getDefaultDisplay().getMetrics(dimensions);

      TSLDouble lat1 = new TSLDouble();
      TSLDouble lon1 = new TSLDouble();
      TSLDouble lat2 = new TSLDouble();
      TSLDouble lon2 = new TSLDouble();

      drawingSurfaceView.drawingSurface().TMCToLatLong(
        currentExtent.getBottomLeft().m_x, currentExtent.getBottomLeft().m_y,
        lat1, lon1);
      drawingSurfaceView.drawingSurface().TMCToLatLong(
        currentExtent.getTopRight().m_x, currentExtent.getTopRight().m_y,
        lat2, lon2);
      terrainDB.displayExtent(
        dimensions.widthPixels, dimensions.heightPixels,
        lon1.getValue(), lat1.getValue(), lon2.getValue(), lat2.getValue());
      }
    }
  }
```

5.11 Direct Import SDK

In order to use the MapLink Direct Import SDK in an application, these steps must occur:

1. The Java, and native Direct Import SDK libraries must be included in the application, see section 4.
2. The MapLink coordinate systems must be loaded: `TSLCoordinateSystemFactory.loadCoordinateSystems`
3. `TSLUtilityFunctions.unlockSupport` must be called with a valid key, in order to unlock the Core SDK runtime which includes a licence for the Direct Import SDK.
4. A `TSLDirectImportDataLayer` object must be created: `TSLDirectImportDataLayerFactory.createDirectImportDataLayer`
5. The `TSLDirectImportDataLayer` must be added to the drawing surface.
6. The output coordinate system of the data layer must be set using `TSLDirectImportDataLayer.setCoordinateSystem`. The data layer will project all data into this coordinate system for display, this is independent of the data's original coordinate system.
7. A scale band must be added to the layer using `addScaleBand` on the data layer object.
8. Create data sets with the map data by initialising a `Collection<TSLDirectImportDataSet>` using the `createDataSets` method on the data layer object.
9. Use the `loadData` function on the data layer object to load the datasets.
    - If the data being loaded is vector data, a `TSLFeatureClassConfig` with rendering attributes must be passed into this function, otherwise the vector data will not be displayed. `TSLFeatureClassConfig` can be initialised with: `TSLFeatureConfigurationFactory.createFeatureClassConfig();`

The `TSLDirectImportCallbacks` class isn't always required but is recommended when using the `TSLDirectImportDataLayer`. If there is data being loaded that is missing a coordinate system or an extent MapLink will request this information via the callback object.

If these callbacks are not implemented, and this information is not available, the data will not be loaded.

For more information please see the API documentation for `TSLDirectImportDataLayer` which is included in the MapLink Pro for Android installation.

A sample is available that showcases the Direct Import SDK and can be found in the Maplink Pro installation directory under '*Samples/BasicMapLinkApplicationDirectImport*'.

DEPLOYMENT OF AN APPLICATION

MapLink Pro uses a significant number of configuration files. These need to be shipped with an application and your application code needs to instruct MapLink on where to find these files before MapLink itself can be used.

5.12 Loading MapLink Configuration Files

The MapLink configuration files are considered to be contents of the `SDK/config` directory from the MapLink installation. These files must be included on the device in some form in a location that is accessible to the application.

Note that the entire configuration directory is not always required. In particular, symbols that are not used in an application can be safely removed. Please contact support@envitia.com if you require additional guidance.

There are two options for including these files in an Android application:

Method	Pros	Cons
Inside the application's assets folder	All necessary configuration data is included inside the application - no external dependencies. No special permissions required in `AndroidManifest.xml`.	Applications (APKs) become larger. The application must decompress the config directory before loading, This may take a significant amount of time the first time the application is run.
Device’s storage medium	Applications (APKs) are smaller. Configuration can be shared across multiple MapLink based applications on a single device.	The location must be known to all applications. If the location changes, each application must be made aware of the new location. Requires external storage read permissions in `AndroidManifest.xml`.

5.12.1 Loading MapLink Configuration Files from the Assets Directory

Android projects can package application-specific data into their `assets` directory. Data within this directory is copied into the APK upon its generation and will be installed on the device along with the application itself. Including the MapLink configuration files in this fashion is easy - simply copy the `SDK/config` directory into the `assets` directory of the application's project. When the application is compiled, the files inside this folder will be packaged into the APK. On non-Windows systems a symlink to the MapLink `config` directory placed inside the application's `assets` folder works equally well.

In order for MapLink to be able to load files from the application's `asset` directory they must first be decompressed into the application’s cache. This may be done using the `TSLMapLinkEnvironment.cacheAsset` method. This class allows the cache to be placed in the application’s internal cache, external cache, or in any other application-specified location.

© 2022 Envitia Ltd

Please see the example source code under `$(MAPLINK_ANDROID_HOME)/Examples` for more information. Configuration caching and loading is performed near the top of `MainActivity.java` in `MainActivity.onCreate`.

5.12.2 Loading MapLink Configuration Files from Internal Storage

MapLink can also load its configuration from a device's internal storage. In this case there is no need to decompress any assets. `TSLDrawingSurfaceFactory.loadStandardConfig` can be called directly with the path to the configuration folder.

It is important to remember that developers are strongly advised to consider the use of the following code to obtain the root of the internal storage medium of a device, as this can be different across devices:

Environment.*getExternalStorageDirectory*().getAbsolutePath();

Applications should also ensure they add the `READ_EXTERNAL_STORAGE` permission to their application's `AndroidManifest.xml` when loading files in this fashion.

5.12.3 Removing unnecessary configuration data

It is usually important to reduce the size of a released application as much as possible. One way in particular to achieve this when using MapLink is to remove any symbols that are not required in the config directory.

The corresponding entries must also be removed from tslsymbolsXXX.dat

5.13 Loading Other MapLink Files From the Asset Directory

Other MapLink data, such as maps created by MapLink Studio, can be loaded from the assets directory in a similar way to MapLink's configuration (using the `TSLMapLinkEnvironment` class).

When loading MapLink maps the application must decompress the folder containing the map, and all subfolders, instead of just decompressing the map itself. Please refer to the example applications for more information.

5.14 MapLink Application Permissions

MapLink generally does not require any special permissions in the application's `AndroidManifest.xml` in order to operate in a disconnected environment, with the exception of requiring `READ_EXTERNAL_STORAGE` when accessing files on a device's storage.

Connected functionality, i.e. MapLink classes that use network connectivity, require the `INTERNET` permission in order to operate. Applications that use the following classes will require this permission:

- `TSLFileLoaderRemote`
- `TSLWMSDataLayer`
- `TSLWMTSDataLayer`

5.15 MapLink Application Feature Declarations

Google encourage applications to declare as part of their `AndroidManifest.xml` features that their application uses. These are used by Google Play to filter applications that will not work on a particular device.

For MapLink based applications, each of the following classes uses the stated features.

- `TSLEGLSurfaceView` or `TSLEGLSurface` requires `<uses-feature android:glEsVersion="0x00020000"/>`

5.16 Proguard

If proguard, or a similar tool is to be used when releasing an application, it is important to exclude the MapLink Java SDKs. This is required as proguard will remove methods that aren't called from Java, but are called from the native components. The following entry should be added to the applications proguard config.

```
-keep class com.envitia.maplink.** { *; }
```

5.17 Licencing

The main functionality of the MapLink CoreSDK, and each additional SDK/runtime component are protected by a licence key, and must be unlocked before use. Licence keys are obtained directly from Envitia support, and are handled separately from the windows licencing functionality.

Licence keys may be full or evaluation keys. Evaluation keys are time-limited, and will display a watermark when used in an application.

All components included in a licence key may be unlocked with a call to `TSLUtilityFunctions.unlockSupport`. Typically this is done in an applications '`onCreate`' method. The '`this`' parameter is the primary activity instance for the application the key is valid for.

```
TSLUtilityFunctions.unlockSupport(TSLKeyedOption.PlatformKey, "key", this);
```

When an application is released, this call must still happen, and as such the key must be embedded in the application. The key should be stored in the application as securely as possible, and the application should be obfuscated using proguard or a similar tool.

Licence keys may be restricted to specific version of MapLink, and to a specific application. Please contact support@envitia.com for more information.

© 2022 Envitia Ltd

6 MAPLINK AND OPENGL

As mentioned previously, MapLink uses OpenGL ES 2.0 for rendering. Developers are encouraged to read the 'OpenGL Drawing Surface' section of the ‘MapLink Pro Developer's Guide’ as this contains substantial information that is directly relevant to the MapLink drawing surface on Android. This section contains additional information that is specific to MapLink for Android.

6.1 Handling Power Events

These normally occur when a device enters deep sleep mode or otherwise pauses an application using OpenGL. When this occurs, all OpenGL resources allocated by the application become invalid and must be recreated. When using the `TSLEGLSurfaceView` MapLink internally detects and handles power events for any OpenGL resources it has created that are accessible from the drawing surface.

If the application has created additional resources (e.g. in a custom data layer), the `TSLRenderCallbackListener.onSurfaceCreated` method can be used to identify when a power event has occurred. This method will be called once during application initialisation when the initial drawing surface is created (provided the listener was registered before this occurs). Each time a power event occurs this method will be called again, at which point an application should recreate any OpenGL resources it uses.

© 2022 Envitia Ltd

7 THREADING

Introducing multi-threading complicates matters as MapLink is not completely thread safe. This is principally to ensure maximum performance. Developers should familiarise themselves with this section and the equivalent section in the 'MapLink Pro Developer's Guide'.

7.1 TSLEGLSurfaceView Rendering Thread

The primary interaction an application with have with multithreading is when using the `TSLEGLSurfaceView`. The Android `GLSurfaceView` class this is based on performs all rendering in a separate thread to the application's UI thread. This means the MapLink drawing surface contained within the surface view resides in this rendering thread and not the application's UI thread.

It is important to note that interactions with the MapLink drawing surface are not thread safe unless explicitly declared as such. Applications should ensure that interactions with the drawing surface occur in the rendering thread. The `Runnable` mechanism provides an easy way of achieving this:

```
/** Code before here runs in the UI thread */
m_surfaceView.queueEvent( new Runnable() {
  public void run()
  {
    /** Code here runs in the rendering thread */
  }
});
/** Code after here runs in the UI thread */
```

It should also be noted that any data layers and their contents in a drawing surface are also considered to be in the rendering thread, and therefore interactions with them from outside this thread must also be thread safe.

7.1.1 OpenGL and Threads

An OpenGL context can only be active in one thread at a time, and each thread can only have one context active. This means that drawing can only occur in the thread associated with its context and attempting to call any functions that result in drawing in another thread will result in errors being generated and no drawing occurring.

Drawing data layers and entities in the MapLink drawing surface creates GPU resources that MapLink associates with the item being drawn. These resources must be freed in the same thread as the OpenGL context that they were created from in order to avoid resource leaks. This can be done either by deleting the object in this thread by calling the `release` method, or using the `releaseResources` method on the `TSLDataLayer` or `TSLEntity`.

The OpenGL context created by the `TSLEGLSurfaceView` and used by the contained drawing surface is bound to a dedicated rendering thread. This context is created by the Android `GLSurfaceView` and therefore should not be moved to a different thread or manually deleted by the application. Doing so will result in undefined behaviour.

© 2022 Envitia Ltd

8 CONFIGURATION DATA FORMATS

The format of the configuration files are defined in the 'MapLink Pro Developer's Guide'. This section lists any differences that are specific to MapLink for Android.

8.1 Fonts

The format of `tslfonts.dat` for Android differs from the format used by MapLink on other platforms.

```
TSLFNT 107              // File ident and format version number
;                              // Field separator on subsequent lines
#This is a comment
I;3;55;symbols/anotherfontfile.dat
#Above is an include declaration to another file.
S;This is a section heading   // Section name for subsequent styles
1;2;DroidSans.ttf             // Font definition
3;2;DroidSans-Bold.ttf        // Font definition
54;2;DroidSerif-Regular.ttf   // Font definition
```

The file declaration and section headings are the same as for other MapLink platforms. Font definitions have the following fields:

- Font ID used for `TSLRenderingAttributeInt.TextFont`
- Reserved. This field must always be `2`.
- Font file to use. This can be a full file path on the system, a path relative to /system/fonts, or a path within the applications assets directory (when loading through a TSLFileLoaderAssetManager). The search order for fonts is:
    1. Full file path.
    2. The applications assets directory if available. See section 5.13 for details of how to setup the assets loader.
    3. File path relative to the /system/fonts.

If additional fonts are added to this file the count on line 3 must be updated accordingly.

If difficulties are experienced when using additional fonts, please check the MapLink error stack. If a font cannot be found an error will be added and will include the paths that were searched to try and locate the font file.

The default `tslfonts.dat` provided by MapLink contains a safe set of default fonts that should exist on most devices. These default fonts are not multilingual. If multiple languages are present in a map, a custom font will need to be used in order to display the text. If the default fonts are used to draw non-latin characters they will often not display the characters, or replace the character with a 'missing glyph' symbol. This symbol is usually a rectangle, sometimes containing a cross or the Unicode codepoint of the character.

© 2022 Envitia Ltd

9 MAPLINK BUILD AND ARCHITECTURE NOTES

MapLink for Android was compiled with the default clang compiler from Google NDK r15c (Clang 5.0.300080). The following architectures are supported:

- Armeabi-v7a
- Arm64-v8a
- x86
- x86_64

Applications can support one or all architectures by including the relevant native libraries.

Note that while arm64-v8a and x86_64 devices are backwards compatible to the 32-bit architectures applications may see large performance gains when using the 64-bit specific libraries. Therefore it is recommended that applications include both sets of libraries when creating applications to run on both architectures.

The following compilation options were specified:

- `APP_STL` is `gnustl_shared`
- `APP_ABI` is `armeabi-v7a arm64-v8a x86 x86_64`
- `LOCAL_CPP_FEATURES` includes both `rtti` and `exceptions`

© 2022 Envitia Ltd

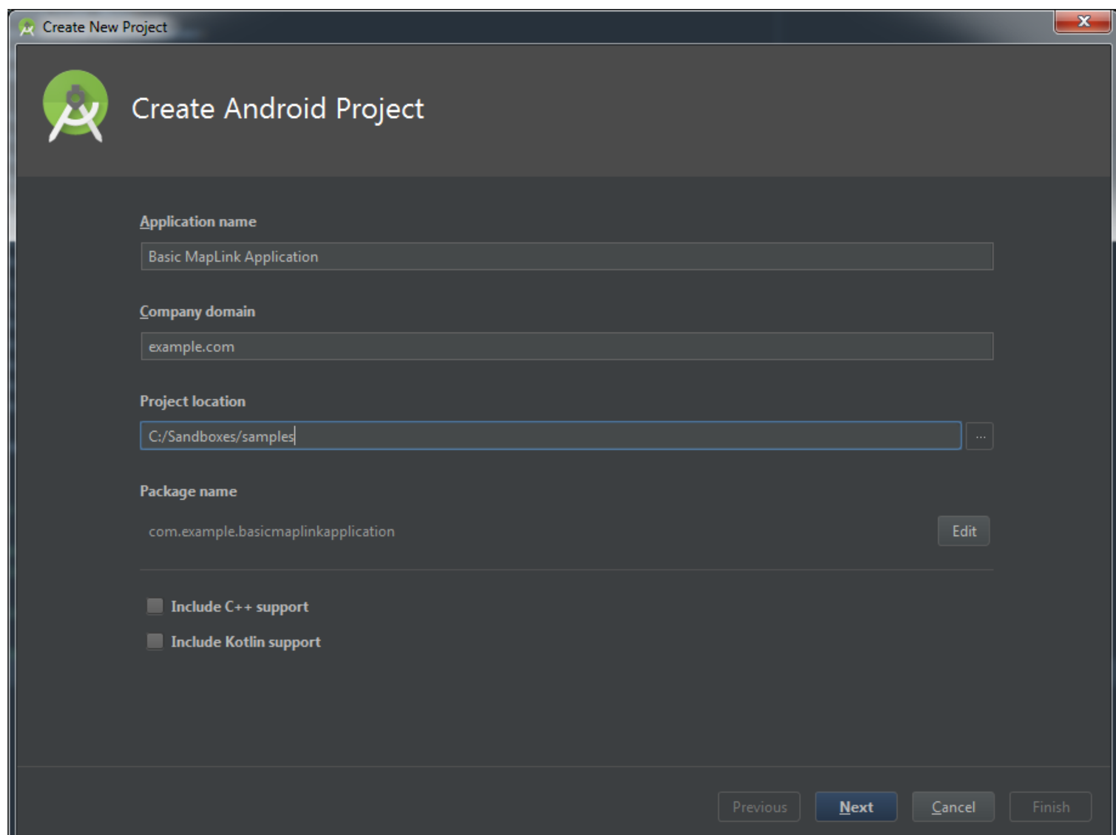
10 SAMPLE APPLICATION WALKTHROUGH - BASIC MAPLINK APPLICATION

This section provides a short guide on how to create a basic MapLink for Android application. The completed code for this sample can be found in the BasicMapLinkApplication sample included in the MapLink installation.

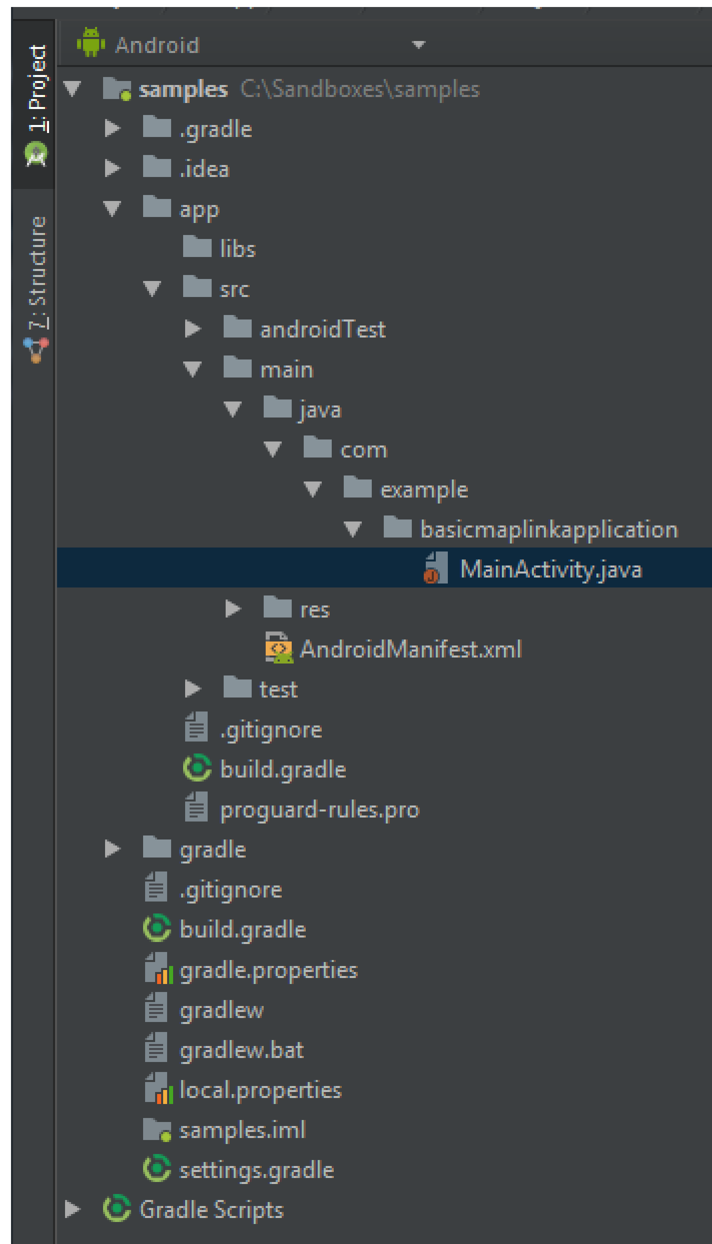
10.1 Creating the Project

First, the project must be created in Android Studio.

1. Click *File > New > New Project…*
2. In the Create Android Project window, provide the application name, company domain, project location, and package name. Press Next.



3. Keep clicking *Next* through the rest of the wizard, leaving the rest of the settings as the default values. At the end of the wizard, Android Studio will show a new project in the *Project Explorer* window:



The Basic MapLink Application project now exists.

10.2 Linking against MapLink for Android

The Android SDK and development tools provide several methods of including third party libraries in an application. The MapLink Android examples achieve this within the `build.gradle` file in order to automate the task.

The location of MapLink will either be determined automatically by `bu`

`ild.gradle` or may be specified by setting the `MAPL_HOME` environment variable.

Please see the MapLink examples for more information.

10.2.1 Java Libraries

```
dependencies {
    // Include maplink.jar in the application
    compile fileTree(dir: maplHome+"/Java", include: ['maplink.jar'])
```

```
}
```

10.2.2 Native Libraries

```
android {
    sourceSets {
        main {
            // Include the native MapLink libraries in the APK
            // The MapLink lib directory has the same structure as the
jniLibs directory
            jniLibs.srcDirs = ['src/main/jniLibs', maplHome+"/lib"]
        }
    }
}
```

10.3 Loading the Native Libraries

Before any of the classes contained in `maplink.jar` can be instantiated and executed, the native libraries must be loaded.

1. Open *MainActivity.java* in the editor.
2. At the top of the class insert the following code:

```
private static boolean nativeLibrariesLoaded = false;
static  {
  nativeLibrariesLoaded = TSLUtilityFunctions.loadNativeLibraries();
}
```

The static initialiser ensures that the native libraries are loaded before any of the functional code is executed. This is required in every application.

It is advisable for the application to verify that the MapLink libraries loaded successfully, as otherwise the application will force close when attempting to use MapLink. At the start of the `onCreate` method of the activity, insert the following code after the call to `super.onCreate`:

```
if( !nativeLibrariesLoaded ) {
  /** The MapLink native libraries failed to load, MapLink cannot be used */
  Toast.makeText(this, "Fatal - Unable to load MapLink native libraries.",
    Toast.LENGTH_LONG).show();
  return;
}
```

This code displays a simple message popup stating that the MapLink native libraries could not be loaded and exits the application's `onCreate` method. If this occurs, go back to section 10.2 and verify that the native libraries are in the correct location in the project.

10.4 Load the MapLink Standard Configuration

In this sample the application will load MapLink's configuration from the application's assets folder. The `config` directory may be copied into the application’s assets folder or copied automatically using `build.gradle`.

Next, insert the following code into the activity's `onCreate` method.

```
TSLMapLinkEnvironment.initialise(this);
String configDir = null;
try {
  configDir = TSLMapLinkEnvironment.instance().cacheAsset("config");
} catch (IOException e) {
  Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
  return;
} catch (TSLMapLinkEnvironmentException e) {
  Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
  return;
}
if (!TSLDrawingSurfaceFactory.loadStandardConfig(configDir))
{
  // Error handling
  Toast.makeText(this, "Fatal - Unable to load MapLink's configuration files.",
    Toast.LENGTH_LONG).show();
  return;
}
```

Here a MapLink asset loader is created and registered as the global resource loader. The MapLink configuration is then loaded from the `config` directory located inside in the application's `asset` folder. If this fails, the activity's `onCreate` method is exited. If this happens, this means that MapLink's `config` directory was not put in the correct location. If the WMS/WMTS datalayers, or the coordinate system classes are to be used, `tsltransforms.dat` should also be loaded using `TSLCoordinateSystemFactory.loadCoordinateSystems`.

10.5 Unlocking licenced components

This sample will require functionality from the MapLink CoreSDK. By default this is not unlocked, please see section 5.17 for additional information.

The following code will unlock the SDK, and all other components that the key is valid for.

```
if(!TSLUtilityFunctions.unlockSupport(
  TSLKeyedOption.PlatformKey, “key”, this)) {
  Toast.makeText(this, "Error - Failed to unlock licenced components.",
    Toast.LENGTH_LONG).show();
  return;
}
```

Note: “`key`” should be replaced with the licence obtained from Envitia support.

If the unlocking fails then no components will have been unlocked and MapLink will not display any data.

10.6 Creating the Activity's User Interface

The user interface for this sample consists of a full-screen MapLink drawing surface.

In the project:

1. Open *res > layout > activity_main.xml* in the graphical layout.
2. Delete the auto-generated *TextView* element.

Unfortunately, `View` objects in external library jars do not appear in the ‘Custom & Library View’ section of the Android Layout Editor, so the MapLink surface view cannot be directly drag-and-dropped into the activity's user interface. Instead, a similar `View` type (in this case a `SurfaceView`) can be used when laying out the activity's user interface, and then the type of the View can be changed to the desired type.

3. Open the Advanced group in the palette window of the graphical layout editor.
4. Draw a `SurfaceView` object on to the activity.
5. Change the View's ID from `surfaceView` to `drawingSurfaceView`.
6. Right click the `SurfaceView` in the layout editor and select *Layout Width > Match Parent* from the context menu. Do the same for the layout height.
7. Switch to the XML view of the layout using the bottom tab.
8. Change the definition of the `SurfaceView` element from this:

```
<SurfaceView android:id="@+id/drawingSurfaceView"
android:layout_width="wrap_content" android:layout_height="wrap_content" … />
```

To this:

```
<com.envitia.maplink.core.drawingsurfaces.TSLEGLSurfaceView
android:id="@+id/drawingSurfaceView" android:layout_width="match_parent"
android:layout_height="match_parent" />
```

The Activity's user interface now looks like this:

Now that the `TSLEGLSurfaceView` exists, it must be initialised when the activity is created. Add a data member for it to the `MainActivity` like so:

```
private TSLEGLSurfaceView drawingSurfaceView;
```

Now add the following code to the bottom of the `onCreate` method (below the call to `TSLDrawingSurfaceFactory.loadStandardConfig` and `TSLUtilityFunctions.unlockSupport`) of the `MainActivity` class to initialise the widget.

If the CoreSDK has not been unlocked and the exception is thrown, the widget will still be initialised. However it will not display anything.

```
drawingSurfaceView = findViewById(R.id.drawingSurfaceView);
try {
    drawingSurfaceView.initialise();
}
catch (TSLComponentNotLicencedException e) {
    Toast.makeText(this, "Fatal - The MapLink CoreSDK has not been unlocked",
Toast.LENGTH_LONG).show();
    return;
}
```

The `TSLEGLSurfaceView` must be notified when the activity is paused and resumed. This is done by overriding the activity's `onPause` and `onResume` methods and forwarding the call on to the `TSLEGLSurfaceView`. Add the following methods to the `MainActivity` class:

```
@Override
protected void onPause() {
  if( drawingSurfaceView != null ) {
    drawingSurfaceView.onPause();
  }
  super.onPause();
}

@Override
protected void onResume() {
  if( drawingSurfaceView != null ) {
    drawingSurfaceView.onResume();
  }
  super.onResume();
}
```

10.7 Add a Map Layer

Now that the drawing surface is in place a map layer is needed to provide content to be drawn. This is achieved by loading a map created by MapLink Studio into a `TSLMapDataLayer` and adding it to the drawing surface. In this sample we will be loading a map from the application's `assets` folder.

Copy the `VMapUK` directory from the maps folder of the MapLink installation into the application's `assets` folder alongside the Maplink `config` directory.

In the Activity's onCreate method add the following code below the call to `m_surfaceView.initialise`:

```
final TSLMapDataLayer mapDataLayer = TSLDataLayerFactory.createMapDataLayer();
String mapFile = null;
try {
    String mapDir =
TSLMapLinkEnvironment.instance().cacheAsset("VMAP0_UK_Map");
    mapFile = mapDir + "/UK&Ireland.map";
} catch (IOException e) {
    Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
    return;
} catch (TSLMapLinkEnvironmentException e) {
    Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
    return;
}

/** Now load the map */
if( !mapDataLayer.loadData( mapFile ) ) {
    Toast.makeText(this, "Fatal - Unable to load map.",
Toast.LENGTH_LONG).show();
    return;
}
```

10.8 Add the Map Data Layer to the Surface View

For the map to appear when the drawing surface draws to the screen the data layer containing it must be added to the drawing surface. As the drawing surface resides in a different thread (the render thread) to the application's user interface, this must be done explicitly in the render thread.

Add the following code just after the `TSLMapDataLayer` initialisation:

```
/** These operations occur in the rendering thread */
drawingSurfaceView.queueEvent( new Runnable() {
    public void run() {
```

```
            /** Get the drawing surface from the view */
            TSLEGLSurface surface = drawingSurfaceView.drawingSurface();

            /** Add the map to the drawing surface */
            surface.addDataLayer(mapDataLayer, "map");

            /** Make the entire map visible */
            surface.reset();

            /** Ask the drawing surface to redraw so the map is visible. */
            drawingSurfaceView.requestRender();
        }
    });
```

The code above will add the map data layer to the drawing surface with the name *map*.

10.9 Adding Interaction Modes

Finally, an interaction mode must be added in order for the user to interact with the map (panning around and pinch zooming). This will be done using one of the premade interaction modes provided with MapLink.

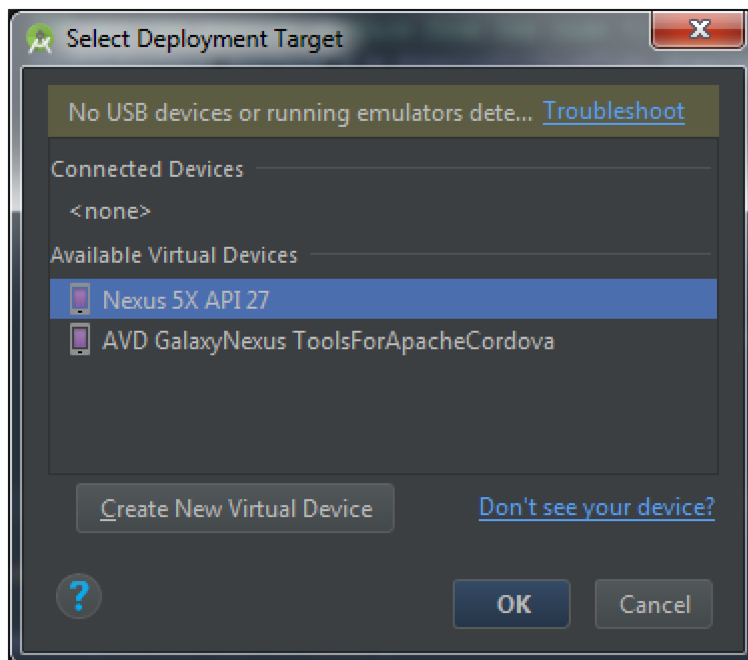
Add the following code below the code just added above:

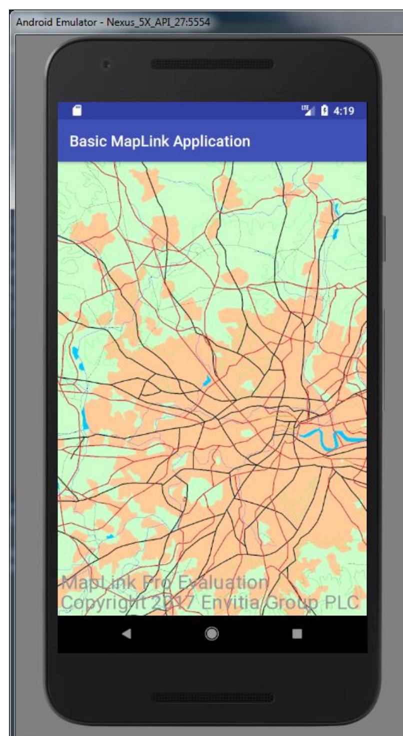
```
TSLPanZoom interactionMode = new TSLPanZoom(getApplication());
drawingSurfaceView.setOnTouchListener(interactionMode);
```

The interaction mode will respond to touch input from the user and adjust the view of the map accordingly. It will instruct the drawing surface to redraw the view as required.

10.10 Launching the application

In Android Studio, click the *Run > Run 'app'* main menu item. The Select Deployment Target window will open. Select the Android device to use. This may be a physical Android device connected to the system or an emulated virtual device.





Above left is how the application should look when it is first launched. Above right is the map data when zoomed over the London area.

11 Basic MapLink Application with APP6A Symbols

The BasicMapLinkApplicationTracks sample is based on the BasicMapLinkApplication sample, and demonstrates a simple method of drawing APP6A symbols moving along random paths on the map. It also shows how to correctly use standalone entities inside a custom data layer in conjunction with the MapLink drawing surface.

The complete code for this sample can be found in the `samples` directory of the MapLink installation.

11.1 Loading the APP6A Symbols

The APP6A Symbols are not loaded as part of the standard configuration, and need loading separately. It is best to do this at the same time as loading the normal MapLink configuration:

```
if (!TSLDrawingSurfaceFactory.loadStandardConfig(configDir)
  || !TSLDrawingSurfaceFactory.setupSymbols(configDir +
"/tslsymbolsAPP6A.dat")) {
    // Error handling
    Toast.makeText(this, "Fatal - Unable to load MapLink's configuration
files.", Toast.LENGTH_LONG).show();
    return;
}
```

11.2 Continuous Rendering on TSLEGLSurfaceView

As this application is designed to update the screen every frame, the `TSLEGLSurfaceView` should be set to update continuously instead of being manually updated by the application:

```
drawingSurfaceView = (TSLEGLSurfaceView)findViewById(R.id.drawingSurfaceView);
drawingSurfaceView.initialise();
drawingSurfaceView.setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
```

When used in this mode the surface view makes callbacks to tell the application when to update. The sample uses this to update the position of the tracks based on how much time has elapsed since the last frame.

In the tracks sample only the `onPreDrawFrame` callback needs to be handled and simply calls the update function on the tracks data layer.

To handle the callbacks the activity must implement `TSLRenderCallbackListener`

```
drawingSurfaceView.setRenderCallbackListener(this);
...
@Override
public void onPreDrawFrame(GL10 gl, TSLEGLSurface surface) {
  tracksDataLayer.updateTracks();
}
```

11.3 Buffered Data Layers

To improve the drawing performance of the application the `TSLMapDataLayer` is set to use layer buffering. This means that the map does not need to be redrawn each frame as long as the view has not changed, which improves performance and battery life.

```
m_drawingSurfaceView.queueEvent( new Runnable() {
  public void run() {
    drawingSurfaceView.drawingSurface().addDataLayer(
      mapDataLayer, "map");
    drawingSurfaceView.drawingSurface().setDataLayerProps(
      "map", TSLProperty.Buffered, 1);
    }
 });
```

11.4 Entity Storage Strategy

As the symbols in the tracks data layer are stored as a hierarchy of entity sets the drawing surface should be set to assign them all to the same resource group for the best performance. The setting of this property is done through the MapLink drawing surface and so must be done on the render thread. Data layer storage strategies are explained in section 12.13.2 of the 'MapLink Pro Developer's Guide'.

```
surface.setLayerStorageStrategy(
  "tracks", TSLOpenGLStorageStrategy.PerEntitySetStrategy);
```

11.5 Implementing the Tracks Data Layer

The `TSLClientCustomDataLayer` class provides an interface to user-defined data layers. In this example it is used to draw standalone entities consisting of APP6A on top of the loaded map.

The `TracksDataLayer` uses the `TrackSymbol` class to manage the position and movement of the symbols. Each `TrackSymbol` consists of the positional data required for the movement calculations and an APP6A symbol which is stored as an entity set.

Each time the surface view is redrawn and calls the `onPreDrawFrame` callback the tracks data layer updates the position of each symbol. This is done by assigning a random heading, distance and speed. Once the symbol has reached its target position the variables are updated, resulting in a continuous random walk. The various parameters used in these calculations are defined at the top of the `TracksDataLayer` class.

A `TSLCustomDataLayer` needs to be created with the `TracksDataLayer` set as the client in order to add the TracksDataLayer to the drawing surface:

```
final TSLCustomDataLayer customLayer =
  TSLDataLayerFactory.createCustomDataLayer();
customLayer.setClientCustomDataLayer(tracksDataLayer);
if(!customLayer.valid()) {
  Toast.makeText(this,
     "The custom data layer is not valid. Check LogCat for more information.",
     Toast.LENGTH_LONG).show();
  return;
}
// On the GL rendering thread, add the tracks data layer to
// the drawing surface with the name "tracks".
drawingSurfaceView.queueEvent( new Runnable() {
  public void run() {
    TSLEGLSurface surface = drawingSurfaceView.drawingSurface();
    surface.addDataLayer(customLayer, "tracks");
  }
});
```

© 2022 Envitia Ltd

11.5.1 Creating APP6A Symbols

The `TSLAPP6AHelper` class provides factory methods for the APP6A symbol objects. When initialising the helper the path to the relevant config file, cached from the assets directory, should be supplied.

```
private TSLAPP6AHelper app6aHelper =
  TSLAAPP6AFactory.createHelper(configDir + "/app6aConfig.csv");
```

To create a useable APP6A Symbol and add it to the top level entity set:

```
private TSLEntitySet entitySet = TSLGeometryFactory.createEntitySet();

TSLAPP6ASymbol symbol = TSLAAPP6AFactory.createSymbol();
// id is an APP6A symbol id, e.g "1.x.2.1.1.2"
app6aHelper.getSymbolFromID(id, symbol);
symbol.hostility(hostility);
symbol.heightType(TSLDimensionUnits.Points);
symbol.height(height);
symbol.setX(randomX); // TMCs
symbol.setY(randomY); // TMCs
entitySet.insert(app6aHelper.getSymbolAsEntitySet(symbol), 0);
```

11.5.2 Drawing the symbols

As the `TSLEntitySet` containing the symbols does not belong to a data layer, it will not be drawn by default. Instead, the tracks data layer must ask MapLink to draw these symbols inside it's `drawLayer` method, through the MapLink `TSLRenderingInterface`:

```
@Override
public boolean drawLayer(
    TSLRenderingInterface renderingInterface,
    final TSLEnvelope extent,
    TSLCustomDataLayerHandler layerHandler) {
  return renderingInterface.drawEntity(entitySet);
}
```

11.5.3 Releasing Entity Resources

When standalone entities or data layers are drawn through the `TSLRenderingInterface` in this fashion, they create OpenGL resources that are tied to the MapLink drawing surface used to draw them. These resources must be deleted from the correct thread, triggered by external circumstances occurring (such as a power event). MapLink will invoke the `releaseResources` method on the tracks data layer in these situations and the tracks data layer must forward this call on to the `TSLEntitySet` it uses for drawing:

```
@Override
public void releaseResources(int surfaceID) {
  entitySet.releaseResources(surfaceID);
}
```

© 2022 Envitia Ltd