Developer User Guide

Document Ref:	**AUM1107- 1.25 \ Draft**
Author:	**Keith Foster**
Date:	**03 January 2025**

ENVITIA

North Heath Lane, Horsham, West Sussex

RH12 5UX, United Kingdom

Tel: 01403 273173

Fax: 01403 273123

Email: envitia@envitia.com

Web: http://www.envitia.com

Table of Contents

List of Figures

1 INTRODUCTION

This document is intended to give developers a guide to designing and implementing solutions using MapLink Pro. It covers all available MapLink SDKs, describing their facilities and how to make the most of them.

The document mostly covers the Windows C++ SDKs. There are few differences between the MapLink Pro Windows and X11 SDK's. The .NET SDK has an almost one-to-one mapping with the C++ SDK so despite code examples being in C++, the same essential principles and steps apply whatever the programming language.

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 Glossary

API Application Programming Interface

BOM Byte Order Mark

DBIF Database Interfaces

DMS Digital Mapping System

DPI Dots per Inch

DDO Dynamic Data Object

DO Display Object

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/

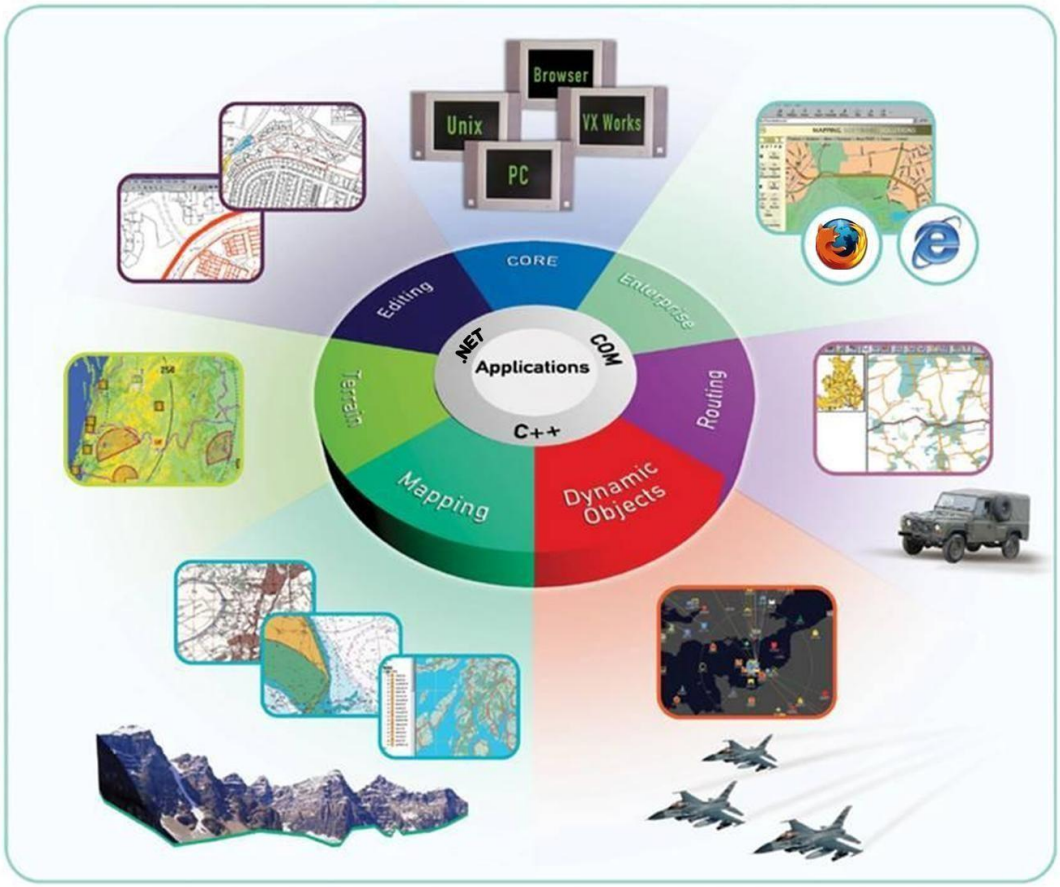
GML Geographic Markup Language http://www.opengeospatial.org/standards/gml

IDE Integrated Development Environment

JPEG	JPEG raster format
Layer	A container that represents a collection of Geometry be it a Map or an Overlay.
MFC	Microsoft Foundation Class
MDI	Multiple Document Interface
STL	C++ Standard Template Library
SDI	Single Document Interface
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 MapLink SDK Components and Concepts

This section gives an overview of each SDK, so that you can determine which of the numerous MapLink Pro components will provide the best facilities for your solution.



2.1 MapLink Studio

MapLink Studio is a sophisticated map processing tool. It allows fusion of a wide range of vector and raster map data with transformation and re-projection into a single integrated picture. As a standalone application, it is usually used off-line to create highly optimised maps. The generated maps are in turn used by user applications built with the suite of MapLink SDK components. MapLink Studio is also a COM Automation Server so can be used on-line and driven by a user application if necessary. More details of MapLink Studio may be found in the other documentation supplied with your MapLink Pro installation, notably the 'MapLink Pro Studio Users Guide'.

2.2 Core SDK

The Core SDK is the basis of all MapLink Pro applications. Like all MapLink SDK's, 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.

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.
- Management of configuration information
- Layering and decluttering of features
- Access to the powerful coordinate system engine for map transformations
- Control of Dynamic Projections for appropriate maps generated by MapLink Studio
- Multi-threaded, progressive map display for smooth, responsive applications
- Double buffering and other optimisation techniques

2.3 OpenGL Drawing Surface

The OpenGL drawing surface allows an application to take advantage of hardware acceleration to enable high performance visualisations on both desktop and mobile platforms.

The drawing surface provided by this SDK may be used as a drop-in replacement for the GDI and X11 drawing surfaces in many situations.

This drawing surface provides additional performance-oriented functionality including the ability to reproject the data in real time.

2.4 Direct Import SDK

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

This SDK provides the ability to reproject 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 like 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 Tracks SDK

The Envitia Tracks SDK provides simple map display and navigation using tracks functionality through a .Net and C++ API.

The Tracks SDK is appropriate for customers who wish to deliver MapLink's highly regarded performance and spatial data visualisation capabilities.

Features include support for:

- Visualisation of real-world entities through Track Display Manager(s).
- Track styling using application-defined symbols or using APP6A and 2525B symbology.
- Track Level of Detail (LOD) visualisations.
- History: The ability to display tracks at any time in the past.
- Track history point trails.
- Direction indicators.
- Visibility of tracks.
- Track selection.

2.6 Dynamic Data Object SDK

The Dynamic Data Object SDK allows developers to create fully dynamic overlays within a MapLink application. Each object within this overlay can have application specific data associated with it through custom derivations of the base class. The architecture splits the real-world Data Object from the visualisation, allowing the same object to be displayed in different ways and in different positions according to application specific rules.

MapLink automatically manages the creation and destruction of the Display Objects that visualise the real-world Data Objects. By default, you have access to the underlying MapLink Rendering Engine used to draw the maps, but if required, you can also override the draw routines allowing any application specific optimisations or features to be employed.

2.7 Editor SDK

The Editor SDK provides facilities for the interactive creation and manipulation of vector overlays. It provides a suite of interactive operations, a customisable operation management interface and components that allow for application specific specialisation of existing operations. In addition, there are components for new application specific operations. The management layer allows all operations to interact with the underlying map or other vector data.

2.8 Terrain SDK

The Terrain SDK provides fast access to layered terrain data that has been prepared through MapLink Studio. The methods on the Terrain Database object allow for query of spot, line and area height information making fast terrain calculations straightforward.

The SDK also allows for generation of contour lines or polygons from an arbitrary data source. Rendering of this contour information is fully controlled by the application.

2.9 MapLink 3D Earth SDK

For software development teams who want to develop and augment high performance three dimensional (3D) displays into their situational awareness mission systems, the MapLink Earth SDK is a software development kit that leverages the highly performant, mature MapLink API and geospatial capabilities of the MapLink software suite with the 3D visualisation capabilities of osgEarth.

Envitia's MapLink Earth SDK provides an API very similar to the legacy MapLink 3D API. Its current capabilities include:

- Applying MapLink's terrain model to 3D model topology;
- Draping MapLink's 2D map layers over terrain;
- MapLink Tracks with optional altitude dimension;
- Billboarded icon symbology.

The MapLink Earth SDK provides the following functionality:

- Draping of MapLink 2D data layers (TSLMapDataLayer, TSLLatLongGridDataLayer, TSLStandardDataLayer) over a 3D terrain;
- Loading of terrain from MapLink Terrain Databases;
- Control of the 3D camera, in the style of the TSL3DCamera class;
- Display of 3D geometry/extruded 2D geometry;
- Visualization of tracked objects as billboarded 2D symbols.

The MapLink Earth SDK is provided as a shared library and header files. The MapLink Earth SDK introduces a new abstract base class (TSLDrawingSurfaceBase) that supports functionality common to both 2D and 3D drawing surfaces. Abstracted functionality includes general layer management principles, support for 2D layers, and other similar operations.

In 2D, manipulations to the map are performed directly on the map – rotation, panning, zooming, etc. In 3D, the canonical implementation is not to manipulate the scene, but instead to alter the viewpoint onto the scene via a camera construct, representing the viewer position and orientation. The camera can pan, tilt and zoom in response to any input desired by the developer.

The ability to chroma-key the 3D scene over a video feed (or any other alternative backdrop) is supported by rendering the scene with a solid, configurable background colour.

Applications can use the SDK to display multiple surfaces using the same shared or discrete data objects, which may be configured with different cameras and visualization parameters.

The SDK supports display of tracked objects as 2D billboards. Rendering can be defined by MapLink vector entities or symbols, MapLink raster symbols and rasterized MapLink military symbols. Track objects provide an elevation dimension to support positioning of the object on or above ground.

2.10 OWSContext SDK

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

2.11 MapLink OGC Services

The MapLink OGC Services SDK should be used to construct one of the OGC Services that MapLink implements. Currently MapLink supports the following OGC Service implementations:

- The Web Map Service (WMS)
- The Web Processing Service (WPS)

2.11.1 The Web Map Service

The MapLink Web Map Service (WMS) provides a framework for serving maps across the internet using standard OGC protocols. The MapLink WMS supports all versions of the OGC Specifications including the latest 1.3.0 version.

2.11.2 The Web Processing Service

The MapLink Web Processing Service (WPS) provides a framework for geospatial web-services in accordance with version 1.0.0 of the OGC specification. The MapLink WPS framework provides the ability to deploy a general purpose "process" via the WPS so that a compatible client can utilise it across a network or the Internet. A "process" could for instance be an operation that converts geospatial data from one format to another, but in truth could be any type of operation that takes zero or more inputs and returns one or more outputs.

2.12 GeoPackage SDK

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.

2.13 Spatial SDK

The Spatial SDK, also known as the LandLink SDK, is formed of two main components.

The first component comprises a set of specialised operations built upon the basic Editor SDK. These operations employ powerful spatial analysis techniques to allow construction of vector data derived from the underlying map. Other custom operations construct specialised graphical primitives especially suited to the Land Registration market.

The second component of the Spatial SDK is a suite of configurable utility classes that provide further map data analysis tools for the automatic construction of Land Registration polygons. Combined with access to address databases such as QuickAddress, this provides a powerful batch processing solution.

The Spatial SDK has also been enhanced with the ability to identify and process self-contained ‘islands’ of changed data, where an island is defined as being a contiguous set of features.

2.14 CADRG Exporter SDK

The CADRG Exporter SDK provides the ability to incorporate the generation of CADRG and CIB within an application using MapLink.

2.15 GML SDK

The GML SDK provides the ability to both read and write OpenGIS® Geography Mark-up Language (GML) version 3.1.1 application schemas and instance data which conform to the Simple Feature Profile Level 0 (SF-0).

2.16 S63 SDK

The S63 SDK allows an OEM to develop an IHO compliant S63 application.

Please refer to the "MapLink S63 SDK Developers Guide".

2.17 Deprecated SDKs

2.17.1 3D SDK

Please note that the deprecated Envitia 3D SDK is no-longer under development. It has been superseded by the MapLink Earth SDK, see Section 2.9.

The 3D SDK extends the capabilities of the Core SDK by allowing users to be immersed in rich geospatial models. The architecture of the 3D SDK is very flexible and parallels that of the existing 2D Core SDK – thus maximising reuse of current MapLink experience. Standard MapLink concepts such as Drawing Surfaces, Data Layers, Terrain Databases and Entities are extended into the 3D space. 2D or 3D Data Layers may be used interchangeably across 2D or 3D Drawing Surfaces – 2D layers may be draped over height fields when displayed on 3D Drawing Surfaces, whilst 3D layers are displayed in plan view on 2D Drawing Surfaces. The Data Layer concept has been extended to allow Terrain Data Layers which are added to the 3D Drawing Surface to provide height information.

3 BASIC MAPLINK APPLICATIONS

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

- Data is loaded into Layers.
- Layers are displayed on Drawing Surfaces.

Of course you'll probably need more detailed information before you design your application, and this section is intended to give you that information.

3.1 Application Architecture

MapLink has almost unparalleled flexibility amongst 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 decision is that you need to add a few extra lines to your code. The benefit is that you don't have to design the rest of your application around MapLink - it will bend to fit into whatever architecture is most suited to your problem domain.

3.1.1 The Document/View Model

The document can be thought of as a MapLink Data Layer, or more likely, a set of Data Layers. The Core SDK contains several specialisations of the base Data Layer, 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 but can also be utilised by the Editor SDK for complex interactive drawing.

The View concept maps directly onto the MapLink `TSLDrawingSurface`. In the same way that a View is a visualisation of a Document, a `TSLDrawingSurface` visualises a set of `TSLDataLayer`'s that have been attached to it. As described above, MapLink is passive, so relevant events should be managed by the application and passed onto MapLink.

For simple applications that merely visualise data, MapLink will only be interested in the initialisation, resize and paint events. These should generally be passed onto the `TSLDrawingSurface`. More complex applications that use the Editor SDK may also need to pass mouse events onto the `TSLEditor` class.

A Data Layer may be displayed in many different Drawing Surfaces, and a Drawing Surface may display many different Data Layers. Each Drawing Surface is independent of the others and may display a different part of the Data Layer or be at a different zoom level.

3.1.2 Error Handling

In keeping with its passive nature, MapLink will not interrupt your application by throwing exceptions or bringing up an error dialog. Instead, it 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. (This class is a thread-safe version of the `TSLErrorStack` class which it supersedes). `TSLErrorStack` is retained for backwards compatibility purposes but its use is discouraged. .NET users have the `TSLNErrorStack` class, which is a wrapper around the `TSLThreadedErrorStack` class, for provision of thread-safe access to the stack.

Error reports on the error stack are encoded in an error number with an associated string to provide further information, such as a filename. Some pre-processor constants and descriptive comments for each error are defined in header files supplied with the MapLink Core SDK. These are the `'*err.h'` files in the include directory. These allow the application to identify the error that has occurred and handle it in a graceful manner.

3.1.3 Error Messages

The Error message strings are stored as a series of message files, which are contained in the `config` directory, all having a '`msg`' file ending. Only the primary message string is contained in these files. Additional information may be generated by the SDK when an error occurs.

The controlling message file is '`allmaplinkerrors.msg`'. This file will only be loaded when the developer requests textual information on the Error.

The '`msg`' files only provide additional information, so if they do not exist, then MapLink will provide basic information. The controlling message file can also be trimmed to reduce the loaded message strings.

Typically you may exclude the '`msg`' files when memory is at a premium.

3.1.4 View and Interaction Modes

Most applications will require some sort of navigation controls such as pan and zoom. To assist with the implementation of these, Envitia supply a suite of useful interaction modes in source code form. The Windows MapLink samples provide the relevant source code, which can be easily incorporated into your own application and modified to suit your requirements.

3.2 Coordinates and Positions

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

Map Units (MU) 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. Thus, 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. The current view of a Drawing Surface is generally 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. These will typically be pixels, but under some circumstances, such as MFC Print Preview, these are modified to represent some other device.

The Drawing Surface and Map Data Layer classes contain methods for conversion between these various coordinate systems.

3.3 Configuration Data

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

Standard configuration files for line-styles, fill-styles, fonts, symbols and colours are provided in the installed MapLink `config` directory. When a style is attached to a Feature Class or to an Entity, it stores the style ID. The run-time rendering engine looks up the ID in the currently loaded style list.

The configuration data is held statically so usually need only be loaded once per application run. The exception to this is the colours file. This is held statically but may need to be reloaded to match the palette for a particular map. This means that if multiple maps are displayed in a single application then they must be created using the same palette.

3.4 Map Display using TSLMapDataLayer

The `TSLMapDataLayer` object manages the display of a Map that has been produced by MapLink Studio. These Maps consist of several configuration files and a set of vector or

raster tiles. The `TSLMapDataLayer` maintains a configurable memory cache of tiles and an optional, configurable disk cache of tiles. The latter is often used for remote or Internet based systems.

As described in the MapLink Studio documentation, a map can consist of several detail layers, each usually representing a particular area of interest at a particular level of detail or zoom with further optimisation available through tiling. A `TSLMapDataLayer` will automatically choose the appropriate detail layer according to configuration parameters set in MapLink Studio. Once a detail layer is chosen, the `TSLMapDataLayer` will manage the loading and caching of relevant tiles. This is the default behaviour. For specific requirements, the Data Layer can be forced to display a particular detail layer as long as the automatic detail layer selection is disabled.

4 UNICODE

MapLink Pro supports Unicode on all supported platforms. This means that you can display multiple languages on a map or layer using MapLink Pro. The data is stored within MapLink as UTF-8.

4.1 Unicode SDK Support

4.1.1 C++ SDKs

The MapLink API uses '`char *`' pointers to pass string information to and from an application. MapLink expects data passed to the C++ API to be UTF-8 or 7-bit ASCII (subset of UTF-8). Data passed back to the application will be UTF-8.

Two helper classes have been provided to help convert text data to and from UTF-8:

- `TSLUTF8Decoder`
- `TSLUTF8Encoder`

In most cases an application may not need to adjust what is passed into MapLink unless the application is being upgraded to Unicode or the user did not use 7-bit ASCII.

For users migrating from a version of MapLink prior to 8.0 please see the section 4.5 for backwards compatibility and workarounds.

4.1.2 .NET SDKs

The .NET SDKs use the Windows concept of Unicode at the MapLink API.

The class `TSLNCoordinateConverter` takes `System::Char` for some of the conversion methods. The data passed in and out in these cases is assumed to be 7-bit ASCII.

4.2 Unicode Geometry

The following Geometry Text primitives are currently supported by MapLink:

- `TSLText / TSLNText`
- `TSLGeodeticText / TSLNGeodeticText`
- `TSL3DText / TSLN3DText`

`TSL3DText`/`TSLN3DText`[1] only supports a subset of 7-bit ASCII.

`TSLText` and `TSLGeodeticText` will display text in multiple languages. The text may contain more than one language and the languages displayed may be left to right and right to left.

[1] Please contact support if this is an issue so that we can gauge the importance of supporting Unicode in the 3D Text primitive.

4.3 Fonts

The font you use is key to the display of text. If the font does not support the language/script then you need to find an alternative font that does. This may happen because not all fonts contain all the necessary glyph entries to display Unicode strings correctly. You can add new fonts to `tslfonts.dat` file.

4.3.1 Freely Available Fonts

The "Google Noto Fonts" use the Apache Licence 2.0. This set of fonts aims to support all the world's languages.

The MapLink configuration file '`tslfonts.dat`' contains references to other freely useable fonts and some which are commercial (commercial/'non-free' fonts are commented out).

4.3.2 Vertical Text Layouts

Unicode vertical text layouts are not currently supported. Any vertical text will be drawn horizontally.

4.3.3 Right to Left Scripts

We support right to left scripts. The alignment of the string is not swapped as it can be in some text editors, therefore the positioning of a left or right aligned text string will be the same for both left to right and right to left strings. You can mix different scripts within a single text item.

4.3.4 Vector Font

The vector font support is limited to 7-bit ASCII on both the GDI and X11 Drawing Surfaces. Vector font drawing is not supported by the 2D OpenGL Drawing Surfaces.

All MapLink drawing surfaces support drawing rotated system text. This negates the need for the Vector font as this was primarily used for drawing rotated text on platforms that could not support drawing of rotated system fonts.

4.4 Filenames and Paths

- All filenames and paths must be encoded as UTF-8.
- MapLink expects the filenames and paths on Windows to be 'long' paths. Passing a 'short' or '8.3' filename may not work correctly. These can be on a local drive 'C:\file.txt' or on a network share '\\server\file.txt'.
- MapLink may also accept 'UNC' paths. These can be on a local drive '\\?\C:\file.txt' or on a network share '\\?\UNC\server\file.txt'.
- Unless an application is using 'UNC' paths elsewhere it is recommended that 'long' filenames be used. Applications do not need to covert paths to 'UNC' format for paths longer than ~260 characters.

- Many sections of the MapLink API support additional formats such as folders, URLs, or datatype-specific identifiers.
- Paths relative to a drive-specific working directory are unsupported, e.g d:file.txt (file.txt relative to the current working directory on drive D).
- MapLink will attempt to convert a path without a drive letter using the current working directory, paths such as '/a/b/c/d.e' will work whereas a path such as 'c/d.e' will be treated as relative to the current working directory.

4.4.1 Path length limitations

On Windows the maximum path length for 'long' file names is ~260 characters. For 'UNC' paths it is 32,767 characters.

The maximum length supported by MapLink is 4096 characters. This applies to both 'long' paths and 'UNC' paths.

4.5 Backwards compatibility

The MapLink Pro backward compatibility (versions prior to MapLink 8.0) has been partially broken with the introduction of Unicode support in the way 8-bit characters are handled.

Prior to support of Unicode users may have relied upon 8-bit character strings being passed through the MapLink Pro API without change. Now this is only true for 7-bit ASCII[2] and 8-bit UTF-8 encoded strings.

You will not be affected by these changes if:

- You only used 7-bit ASCII strings.
- You used the .NET SDKs.
- You use a non-Windows platform (these are usually UTF-8 by default).

In order to support Unicode with MapLink Pro we have had to break the following at the API:

- Filename and path names have to be long filename/paths on Windows
- Text has to be UTF-8. We no longer support Text being passed to MapLink as defined by the System Code page.
- All text is assumed to be UTF-8.

To minimise the impact of this change we will read and convert all text from files generated by MapLink versions prior to 8.0 on load to UTF-8 where possible. The default conversion assumes the files contain text in the System Code Page on Windows, and CP-1252 on other platforms.

[2] 7-bit ASCII is a subset of UTF-8

4.5.1 Workarounds

4.5.2 Text

Text is now expected to be 7-bit ASCII or UTF-8.

When MapLink Pro reads files generated by versions prior to 8.0 we process the string as follows:

```
Read text
If the text is not UTF-8
Convert to UTF-8 using the System Code Page
```

In most cases this should be completely transparent to the application. If you experience problems with text not being correctly converted, you can override the Code Page used for the application using the following unsupported[3] methods:

- `void TSLifstream::legacySetEncodingOverride(TSLTextEncoding encoding[4]);`
- `TSLTextEncoding TSLifstream::legacyGetEncodingOverride();`
- `void TSLofstream::legacySetEncodingOverride(TSLTextEncoding encoding[5]);`
- `TSLTextEncoding TSLofstream::legacyGetEncodingOverride();`

Setting a Code Page encoding to use will affect **all** text reading or writing.

Note:

Because we convert from UTF-8 to Multi-Byte for saving older version of the MapLink file formats some languages may not convert well. The more complex the script, such as Arabic, the less likely this will work.

If you use this approach, all data stored in the MapLink control files must be 7-bit ASCII or in the Code Page that has been set. You cannot mix text in multiple Code Pages.

4.5.3 Text File Formats

The MapLink configuration files and control files are principally text based. If you have modified the MapLink configuration files from a version of MapLink prior to 8.0 and you wish to use these with MapLink 8.0 or later you may experience problems.

In the case of configuration files you should open the file in an editor such as Notepad++ and convert the file to UTF-8 without BOM (Byte Order Mark) and save the file. The version at the top of the file should be updated to the latest version number for that file format.

---

[3] The methods will be removed in a future release of MapLink. If you use these methods please let us know why so that we can assess the impact of removal.

[4] Only a limited set of Code Pages are listed in this enum. Please contact support@envitia.com if you need to use a Code Page not listed.

[5] Only a limited set of Code Pages are listed in this enum. Please contact support@envitia.com if you need to use a Code Page not listed.

You should not update layer or MapLink Studio configuration files are these contain multiple objects with each possibly at a different version.

4.6 Unicode FAQ

Why have you chosen to use UTF-8 as the Unicode representation for MapLink Pro?

- UTF-8 is cross platform.
- UTF-8 is compatible with 7-bit ASCII.
- The string terminator is still '\0'.

This means that significantly less needs to be changed internally to support Unicode text within both MapLink and end user applications across all the platforms supported.

Why did you not use `wchar_t`?

- `wchar_t` can be any size from 1 byte to 4 bytes depending on the platform being used. The internal encoding of `wchar_t` is also platform dependent.
- `wchar_t` is not compatible with 7-bit ASCII.
- The string terminator might not be '\0'.
- From The Unicode Standard, chapter 5:
    - The width of `wchar_t` is compiler-specific and can be as small as 8 bits. Consequently, programs that need to be portable across any C or C++ compiler should not use `wchar_t` for storing Unicode text.

Why did you not use UCS2?

- UCS2 is only the default encoding for Unicode text on Windows.
- UCS2 is not backwards compatible with 7-bit ASCII.

Where can I find out more about UTF-8?

http://www.utf8everywhere.org/

5 MAPLINK AND YOUR DEVELOPMENT ENVIRONMENT

5.1 Library Usage and Configuration

As of version 11.1, MapLink is no longer supplied with Debug or 32-bit libraries. Therefore, your application's build should link against the Release Mode libraries in all configurations.

MapLink64.dll

Release mode, DLL version.
Uses Multithreaded DLL C++ run-time library.

Requires TTLDLL preprocessor directive.

Refer to the document "MapLink Pro X.Y: Deployment of End User Applications" for a list of run-time dependencies when redistributing. Where X.Y is the version of MapLink you are deploying.

6 Deployment of End User Application

Please refer to the "MapLink Pro X.Y: Deployment of End User Applications" for detailed information for deploying an application, including copyrights, deployment restrictions and required DLLs.

The following sections are an overview of the necessary code changes for deployment.

6.1 Configuration Files

MapLink Pro loads its necessary configuration files from the '`<MapLink Installation>\config`' directory, usually through a call to `TSLDrawingSurface::loadStandardConfig`. When deploying an application based upon MapLink, a copy of this folder must be shipped along with the application.

The MapLink Pro installer adds a reference to the system registry to allow the MapLink libraries to locate the `config` directory at runtime. Envitia do not recommend however, that this registry key or any MapLink environment variables are used when deploying applications based upon MapLink.

Therefore as the MapLink libraries will not know the new location of this directory on the deployment machine's file system, calls to various MapLink methods will need to be changed to be passed the location of the `config` directory. The following table lists the current method calls which will need to be updated, depending upon the technology being used:

Note: If an application to be deployed does not use a method mentioned, then that method may be ignored.

6.2 C++

Method
`TSLDrawingSurface::loadStandardConfig`
`TSLDrawingSurface::setupColours` - Pass the location of the `tslcolours.dat` file that the config directory contains.
`TSLDrawingSurface::setupFillStyles` - Pass the location of the `tslfillstyles.dat` file that the config directory contains.
`TSLDrawingSurface::setupFonts` - Pass the location of the `tslfonts.dat` file that the `config` directory contains.
`TSLDrawingSurface::setupLineStyles` - Pass the location of the `tsllinestyles.dat` file that the config directory contains.
`TSLDrawingSurface::setupSymbols` - Pass the location of the `tslsymbols.dat` file that the `config` directory contains.
`TSLCoordinateSystem::loadCoordinateSystems` - Pass the location of the `tsltransforms.dat` file that the `config` directory contains.

`TSLAPP6AHelper::TSLAPP6AHelper`
`TSLAPP6AHelper::setDefaultConfigPath`
`TSL3DDrawingSurface::loadStandardConfig`
`TSL3DDrawingSurface::setupModels` - Pass the location of the `tslmodels.dat` file that the `config` directory contains.
`TSLUtilityFunctions::setMapLinkHome` – set the directory that contains the MapLink config **directory.**

6.3 .NET

Method
`TSLNDrawingSurface::loadStandardConfig`
`TSLNDrawingSurface::setupColours` - Pass the location of the `tslcolours.dat` file that the `config` directory contains.
`TSLNDrawingSurface::setupFillStyles` - Pass the location of the `tslfillstyles.dat` file that the `config` directory contains.
`TSLNDrawingSurface::setupFonts` - Pass the location of the `tslfonts.dat` file that the `config` directory contains.
`TSLNDrawingSurface::setupLineStyles` - Pass the location of the `tsllinestyles.dat` file that the `config` directory contains.
`TSLNDrawingSurface::setupSymbols` - Pass the location of the `tslsymbols.dat` file that the `config` directory contains.
`TSLNCoordinateSystem::loadCoordinateSystems` - Pass the location of the `tsltransforms.dat` file that the `config` directory contains.
`TSLNAPP6AHelper::TSLAPP6AHelper`
`TSLNAPP6AHelper::setDefaultConfigPath`
`TSLN3DDrawingSurface::loadStandardConfig`
`TSLN3DDrawingSurface::setupModels` - Pass the location of the `tslmodels.dat` file that the `config` directory contains.
`TSLNUtilityFunctions::setMapLinkHome` – set the directory that contains the MapLink config **directory.**

7 SAMPLES

MapLink Pro includes numerous samples to help you with starting development with an SDK.

Samples are provided for .NET and C++.

While many of the samples are platform specific MapLink Pro itself can be used on both Windows and X11 platforms (Linux and Solaris. Other platforms can be ported to so please contact your Sales representative to discuss).

The samples for all platforms can be used to help inform you as to how to use the SDKs on all platforms.

There are several classes that you need to swap out between platforms that relate to Drawing technology (Drawing Surfaces).

If an SDK is not supported on your platform, please contact your Sales representative to discuss.

7.1 Qt Samples

This section assumes that the reader is familiarly with developing with Qt.

The Qt samples do not have pre-configured Visual Studio or makefiles. Instead we ship QMake project '.pro' (See QMake Project) files from which you need to generate the necessary build files.

The following is a summary of the necessary steps to build the samples on Windows.

- Install Qt or build Qt yourself.
  - You should match the Qt version with the version of Visual Studio you are using.
  - The samples are configured to build in the MapLink Pro installation. If they are moved the pro/pri files will need to be updated.
- Open the 'Qt' Command prompt.
  - Change directory to the sample.
- `set QMAKESPEC=win-64msvc2022`
  - `qmake -t vcapp`
- Load the generated `.vcxproj` into Visual Studio.
- Depending on the version of the Visual Studio you are using you may be prompted to upgrade the project. Accept this upgrade.
- Set the Path to the Qt and MapLink libraries if necessary. By default, the samples are set to be built from the directory they are installed to.
- Build the solution.
- Run the example.

On X11 systems QMake will generate makefiles suitable for use with GNU Make. The MapLink installation will be located using the `MAPL_HOME` environmental variable, which can be set automatically using the `mapl_init` or `mapl_init_bash` scripts.

Additional Notes:

- If using QtCreator load the .pro file as a project into QtCreator.
- If you move the samples you need to modify the .pro file before you create the project to point to the MapLink headers and lib files.

8 WALKTHROUGH 1 - YOUR FIRST MAPLINK APPLICATION

This section guides you through constructing a simple MapLink application from the ground up. By the end, you should have an application that can load and display a map generated from MapLink Studio and can correctly handle expose and resize events.

The example is based on MFC and the C++ SDK, but the same steps apply on X11 targets and with the other MapLink SDK's.

8.1 Skeleton Application

The starting point for this is an MFC Application Wizard generated executable. It can be either an SDI or MDI application. The example code here will be based upon an MDI application.

Use the standard MFC Application Wizard to generate your skeleton application. The example application here is called "Hello Globe".

8.2 Configure Project Properties

Once created, build your skeleton application to ensure it compiles and links. You then need to set up the Project Properties according to the version of the MapLink libraries you wish to use. These are briefly described in section 5.1 and in the "MapLink Pro: Installation and Upgrade Notes".

Make the following checks and modifications to the Project Properties:

- Under the C/C++, General category, add the MapLink include directory as an additional include path, e.g. "`C:\Program Files\Envitia\MapLink pro\X.Y\include`"
- Under the C/C++, Pre-processor category: add `TTLDLL`

Then check the following settings dependent on which configuration you are using.

Note: '**`X.Y`**' refers to the MapLink version you are using.

- Under the Linker, General category: add the MapLink lib64 directory as additional library path e.g. "`C:\Program Files\Envitia\MapLink Pro\X.Y\lib64`"
- Under the Linker, Input category: add `MapLink64.lib`.
- Under the C/C++, Code Generation, select run-time library "`Multi-Threaded Debug DLL`" for Debug configuration or "`Multi-Threaded DLL`" for Release configuration.

Add `#include "MapLink.h"` to relevant files. In this example, just add it into `stdafx.h` to keep things simple.

8.3 API Types

Before we start the first walkthrough example, we should mention that MapLink Pro has a few types to help with clearly identifying what a variable is used for and for portability (cross platform and 64-bit).

The types are defined in `tslplatformtypes.h` and `tslatomic.h`.

The general types used are as follows:

General API Type	Meaning
TSLTMC	TMC value.
TSLFeatureID	Feature ID
TSLVersion	Version related
TSLPropertyValue	Property value.
TSLStyleID	Linestyle ID, fill, text, colour etc.
TSLDeviceUnits	Pixels or surface specific device units
TSLFileLength	64-bit signed integer to store a file length in.
TSLFilePosition	64-bit signed integer to store a file position in.
TSLFileOffset	64-bit signed integer to store a file offset in.
TSLTimeType	64-bit time value.

The types used to return OS specific drawing handles are as follows:

OS Specific API Type	Meaning
TSLDeviceContext	Windows 'HDC'.
TSLWindowHandle	Windows 'HWND'.
TSLBitmapHandle	Windows 'HBITMAP', X11 specific structure (defined below).
TSLDrawableHandle	X11 'Drawable'.
TSLVisualHandle	X11 'Visual *'.
TSLColourmapHandle	X11 'Colormap'.
TSLScreenHandle	X11 'Screen *'.
TSLDisplayHandle	X11 'Display *'.

For example; If you pass a specific OS type to a method or you are querying a method which returns an OS specific type as defined above you may need to cast the result or argument to the type expected.

8.4 Initialisation and Clean Up

The configuration files for MapLink are usually only loaded once per execution run using static methods of `TSLDrawingSurface`. In an MFC application, these are normally loaded during the `InitInstance` method of the Application object. The simplest way is to tell MapLink to load all standard configuration files from a particular directory. If no directory is specified, then MapLink will assume that a full MapLink installation has taken place and will attempt to load from there.

In the method InitInstance method of the App object, add a call to `TSLDrawingSurface::loadStandardConfig`. This should be done before the Document Template is instantiated.

You should be careful to check for, and report errors at this stage by using the methods supplied on the `TSLThreadedErrorStack` utility class.

```
// Initialise MapLink configuration files.
const char * configDirPath = 0 ; // Replace if deployed

TSLThreadedErrorStack::clear() ;
TSLDrawingSurface::loadStandardConfig( configDirPath ) ;

// CHeck to see if an errors occured.
TSLSimpleString msg( "" );
bool anyErrors =
        TSLThreadedErrorStack::errorString( msg,
                                "Initialisation Errors : \n" ) ;
if ( anyErrors )
{
  AfxMessageBox( (LPCTSTR)TSLUTF8Decoder(msg), MB_OK ) ;
  exit( 0 ) ;
}
```

When your application is deployed, make `configDirPath` variable point to the location of your applications copy of the MapLink `config` directory.

Once MapLink has been initialised, it needs to be cleaned up when the application exits, otherwise Visual Studio will report numerous “leaks” which are in fact memory currently in use when the application exits. This should be done in the `ExitInstance` method of the App class. You will need to use the class Properties Overrides to add this method since the MFC Application Wizard doesn’t add it by default. Alternatively, in Single Document applications, it may be called in the destructor of the View or Document class.

Use `Properties`, `Overrides` to create an `ExitInstance` method on the App object. In this method, call MapLink to cleanup the configuration file load.

```
TSLDrawingSurface::cleanup( ) ;
```

8.5 Managing the Document

In terms of the Document/View architecture, the Document contains one or more MapLink Data Layers. For the purposes of this example application, we shall restrict this to a single `TSLMapDataLayer`.

In the private section of the Document, declare a pointer to a `TSLMapDataLayer` object. This should be initialised to `NULL` in the Document constructor.

Use Properties, Overrides to create an `OnOpenDocument` handler and in this method, instantiate a Data Layer and load the map file ensuring that you check for errors.

```
BOOL CHelloGlobeDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
  if (!CDocument::OnOpenDocument(lpszPathName))
    return FALSE;

  m_mapDataLayer = new TSLMapDataLayer() ;

  if ( !m_mapDataLayer->loadData( lpszPathName ) )
  {
    TSLSimpleString msg( "" );
    bool anyErrors = TSLThreadedErrorStack::errorString( msg,
    "Cannot load map : \n" ) ;
    if ( anyErrors )
      AfxMessageBox( msg, MB_OK ) ;

    m_mapDataLayer->destroy() ;
    m_mapDataLayer = NULL ;

    return FALSE ;
  }

  return TRUE;
}
```

Of course, you should also destroy the Data Layer once it is finished with.

Use Properties, Overrides to override the `DeleteContents` method and in here, destroy the Map Data Layer

```
void CHelloGlobeDoc::DeleteContents()
{
  if ( m_mapDataLayer )
  {
    m_mapDataLayer->destroy() ;
    m_mapDataLayer = NULL ;
  }
  CDocument::DeleteContents();
}
```

8.6 Managing the View

In terms of the Document/View architecture, the View contains an instance of a `TSLDrawingSurface` derived object – `TSLNTSurface` on Windows platforms, `TSLMotifSurface` (for historical reasons this surface has Motif in its name however the drawing surface only uses X11 client libraries such as Xft, XRender and Xlib) on X11 platforms. This is the only significant platform specific difference. In an MFC application, this is usually instantiated in the `OnInitialUpdate` method since the associated window doesn't exist in the `OnCreate` event or in the View constructor.

In the private section of the View, declare a pointer to a `TSLNTSurface` object. This should be initialised to `NULL` in the View constructor.

Use Properties, Overrides to create an `OnInitialUpdate` handler and in this method, check to see if a Drawing Surface exists and create one if necessary. You should also tell MapLink about the default size of the window. In the destructor of the View, delete the Drawing Surface if it exists.

```
void CHelloGlobeView::OnInitialUpdate()
{
  CView::OnInitialUpdate();

  if ( !m_drawingSurface )
  {
    m_drawingSurface = new TSLNTSurface( m_hWnd, false ) ;

    RECT cr ;
    GetClientRect( &cr ) ;
    m_drawingSurface->wndResize( cr.left,  cr.top,
         cr.right, cr.bottom, false ) ;
  }
}

CHelloGlobeView::~CHelloGlobeView()
{
  if ( m_drawingSurface )
  {
    delete m_drawingSurface ;
    m_drawingSurface = NULL ;
  }
}
```

8.7 Binding Layers and Drawing Surfaces

Once both Document and View are ready and available, you need to attach the Data Layers to the Drawing Surface so that MapLink can display it.

The recommended approach to this is to create an `addToSurface` method on the Document, which calls the underlying MapLink routines to add the Documents Data Layers to the Views Drawing Surface. This structure avoids the View knowing the contents of

Document in any detail and is equally applicable to both Single and Multiple Document Interfaces.

The `addToSurface` method should be called in the `OnInitialUpdate` method of the View, just after the Drawing Surface has been created. In MFC applications, it is not usually necessary to have an equivalent `deleteFromSurface` method since MFC calls `DeleteContents` instead. If you are adding more than one Data Layer to the Drawing Surface, each must have a unique name.

Create a public `addToSurface` method in the Document that takes a `TSLDrawingSurface` pointer as a parameter. In this, add the Document's Data Layer to the specified Drawing Surface.

```
bool CHelloGlobeDoc::addToSurface(TSLDrawingSurface *drawingSurface)
{
  if ( !m_mapDataLayer || !drawingSurface )
    return false ;

  return drawingSurface->addDataLayer( m_mapDataLayer, "map" ) ;
}
```

Call this method in the View's `OnInitialUpdate` method, after the Drawing Surface has been created. At this point, it is also appropriate to define the initial visible map area. Here we call the reset method to display the entire map.

```
  if ( GetDocument()->addToSurface( m_drawingSurface ) )
 m_drawingSurface->reset( false ) ;
```

Note that MapLink automatically takes care of Data Layer and Drawing Surface separation when either is destroyed.

8.8 Handling Resize Events

Since MapLink is passive, the application needs to handle relevant events and pass the information onto MapLink. Most applications will only need to handle the window resize and expose or paint events.

After handling a resize event, Windows or X will usually post a paint message so there is no need to force a redraw in the resize handler. Just changing the window size may distort the aspect ratio of the display, so MapLink can automatically adjust the visible map area to be in sympathy with the aspect ratio of the window. This optional behaviour allows an anchor point to be specified, which is kept at the same place when updating the visible map area.

Use Properties, Messages to create a `WM_SIZE` handler on the View class since it is not there by default. In this method, check to see if a Drawing Surface exists and if so, pass the new corners of the window to the Drawing Surface using the `wndResize` method. This example will also inhibit an automatic redraw and ask MapLink to maintain the aspect ratio locking the top left corner of the visible map area.

```
void CHelloGlobeView::OnSize(UINT nType, int cx, int cy)
{
  CView::OnSize(nType, cx, cy);

  if ( m_drawingSurface )
  {
    m_drawingSurface->wndResize( 0, 0, cx, cy, false,
                                 TSLResizeActionMaintainTopLeft );
  }
}
```

8.9 Handling Paint Events

A paint event can be triggered for many reasons, some of which will only want to redraw part of the window. Under these circumstances, Windows will set up a Clip Box to define the part that needs redrawing. To improve performance, it is best to only redraw that part. It is most efficient to pass the required Device Unit extent to the Drawing Surface.

In the `OnDraw` method of the View, query the required redraw area and pass it to the Drawing Surface, asking MapLink to clear the background first.

```
void CHelloGlobeView::OnDraw(CDC* pDC)
{
  if ( m_drawingSurface )
  {
    RECT rect ;

    if ( pDC->GetClipBox( &rect ) == NULLREGION )
      GetClientRect( &rect ) ;

    m_drawingSurface->drawDU( rect.left, rect.bottom,
                              rect.right, rect.top, true ) ;
  }
}
```

Now build the program, run it and load one of the sample maps.

8.10 Reducing Flicker and Improving Performance

So far, the application is not making use of MapLink performance optimisations and the display will appear to flicker when it is redrawn. There are two reasons for this. Firstly, MapLink is drawing directly to the window. Secondly, both MapLink and Windows are clearing the display prior to the redraw. In depth discussion of these problems and their solutions may be found in section 12.5. In the meantime, here are a couple of quick fixes to reduce your eyestrain!

To solve the first issue, a single method call should be added when the Drawing Surface is created to make it buffered. This will also improve performance on expose events that are not due to the visible map area changing.

In the `OnInitialUpdate` method of the View, add the following call immediately after the creation of the Drawing Surface.

```
m_drawingSurface->setOption( TSLOptionDoubleBuffered, true ) ;
```

To solve the second issue, you should inhibit Windows from clearing the window.

Use Properties, Messages to add a View handler for the `WM_ERASEBKGND` message. Return `TRUE` from this method to indicate to windows that the application will erase the background.

```
BOOL CHelloGlobeView::OnEraseBkgnd(CDC* pDC)
{
  return TRUE ;
}
```

The inhibition of the `WM_ERASEBKGND` message is appropriate since MapLink is drawing to the entire window. If MapLink were drawing to only part of the window then it may be necessary for the application to erase the areas that MapLink is not rendering into.

9 WALKTHROUGH 2 - MODIFYING THE VISIBLE AREA

Congratulations! You have now created your first MapLink application. As applications go though, it's not the most useful since you are restricted to looking at the whole of the map area. The next step is to add some user interaction and thus look at any area of the map.

9.1 Defining and Implementing an Interaction Model

Firstly you must decide on your interaction model. This is the combination of menu selections and mouse actions. With the sample applications, Envitia supplies the source code for a pre-built interaction model. This varies according to the specific application since some have more complex requirements such as editing. These usually require switching modes via menu selections or toolbar buttons.

For the purposes of this example, we will now implement the following interaction model, which does not require any additional menu items.

- Left button click: Pan to clicked point and zoom in by 25%
- Right button click: Zoom out by 25%
- Left button drag: Zoom in to chosen rectangle
- Right button drag: Grab and move the current view with the mouse

We shall define a click as a button press/release cycle where the cursor does not move more than 3 pixels between press/release. A drag is a press/move/release action where the movement is more than 3 pixels.

As an extension, we will also add some handling for the mouse wheel, should one be installed. You should generally be careful not to create an interaction mechanism that is totally dependent upon the mouse wheel since the user may not have such hardware. There are also various driver issues regarding wheel support that are discussed later.

- Wheel spin: Zoom in and out
- Wheel press (or middle button click with 3 button mouse): Pan to clicked point

For deployed applications, it is recommended that the `TSLInteractionModes` are used. These examples are merely to show how custom interactions could be used and to promote understanding of what goes on "under the bonnet" of MapLink.

9.1.1 Adding Simple Zoom/Pan Handlers

These handlers should be added to the View since each Drawing Surface is independent even if attached to the same Data Layers. Note that all the view manipulation methods on the `TSLDrawingSurface` return true if they were successful and false if unsuccessful. A failure usually indicates that the Coordinate Space limits were reached. It is usually more efficient to inhibit the automatic redraw in these methods and control the redraw by invalidating the window. This is especially true when applying echo styles, for example with zoom to rectangle.

Use Properties, Messages to add handlers to the View for the `WM_LBUTTONUP` and `WM_RBUTTONUP` events.

In the `OnLButtonUp` method, check that a Drawing Surface has been created and if so convert the mouse position to a User Unit position. This position should be used as a parameter to the `TSLDrawingSurface::pan` method. If both pan and zoom were successful, then invalidate the window rectangle to force a redraw.

```
void CHelloGlobeView::OnLButtonUp(UINT nFlags, CPoint point)
{
   if ( m_drawingSurface )
   {
     double uux, uuy ;
     if (m_drawingSurface->DUToUU( point.x, point.y, &uux, &uuy ) )
     {
       if ( m_drawingSurface->pan( uux, uuy, false ) )
       {
         if ( m_drawingSurface->zoom( 25, true, false ) )
          InvalidateRect( 0, FALSE ) ;
       }
     }
   }
   CView::OnLButtonUp(nFlags, point);
 }
```

In the `OnRButtonUp` method, check that a Drawing Surface has been created and if so, simply zoom out.

```
void CHelloGlobeView::OnRButtonUp(UINT nFlags, CPoint point)
{
  if ( m_drawingSurface )
  {
    if ( m_drawingSurface->zoom( 25, false, false ) )
     InvalidateRect( 0, FALSE ) ;
  }
  CView::OnRButtonUp(nFlags, point);
}
```

9.1.2 Zoom to Rectangle

For this interaction we need to remember the position that the user first pressed the left button, and when they release it compare the press and release positions. If they are more than 3 pixels apart then we can assume that they have dragged a rectangle, convert the press and release mouse positions to User Units and ask the `TSLDrawingSurface` to display that area. Of course, the aspect ratio of the selected rectangle may not match the aspect ratio of the window. To cope with such situations, the `TSLDrawingSurface::resize` method has a parameter to indicate that MapLink should adjust the specified rectangle to match the window aspect ratio. If the aspect ratios are mismatched, then MapLink will attempt to ensure that the entirety of the specified rectangle is displayed.

Add a member variable to hold the pressed mouse location: `CPoint m_lmb`. Use Properties, Messages to add a handler to the View for the `WM_LBUTTONDOWN` event.

In the `OnLButtonDown` method, store the mouse position.

```
void CHelloGlobeView::OnLButtonDown(UINT nFlags, CPoint point)
{
  m_lmb = point ;
  CView::OnLButtonDown(nFlags, point);
}
```

Modify the `OnLButtonUp` method after the Drawing Surface has been validated.

```
if ( abs( point.x - m_lmb.x ) <= 3 && abs( point.y - m_lmb.y ) <= 3 )
{
  // Pan to point and zoom
  double x, y ;
  if ( m_drawingSurface->DUToUU( point.x, point.y, &x, &y ) )
  {
    if ( m_drawingSurface->pan( x, y, false ) )
    {
      if ( m_drawingSurface->zoom( 25, true, false ) )
        InvalidateRect( 0, FALSE ) ;
    }
  }
}
else
{
  // Zoom to rectangle
  double x1, y1, x2, y2 ;
  if (   m_drawingSurface->DUToUU( point.x, point.y, &x1, &y1 )
      && m_drawingSurface->DUToUU( m_lmb.x, m_lmb.y, &x2, &y2 ) )
  {
    if ( m_drawingSurface->resize( x1, y1, x2, y2, false, true ) )
      InvalidateRect( 0, FALSE ) ;
  }
}
```

The `TSLViewMode` classes supplied with the MapLink SDK samples have a fully functional "Zoom to Rectangle" mode, including echo of the rubber-band rectangle. For the purposes of this simple introduction, we shall ignore the echo rectangle. For information about the echo modes, please see the "mfc" sample.

9.1.3 Grab Pan

For this interaction we need to remember the position that the user first pressed the right button and when the mouse moves, compare the mouse position with the press. If they are more than 3 pixels apart then we can assume that they have dragged the cursor and pan the map appropriately. This will also need a slight modification to the right button release handler to inhibit the zoom out if a grab has occurred.

Add a member variable to hold the pressed mouse location: `CPoint m_rmb` and also to hold the last grabbed position: `CPoint m_lastGrabPoint`. This variable will be used to calculate delta offsets for the pan. We also need flags to indicate whether a grab has occurred and whether one should be checked for: `bool m_grabbed, m_checkForGrab`.

In the View constructor, initialise the boolean variables

```
CHelloGlobeView::CHelloGlobeView()
  : m_drawingSurface(NULL), m_grabbed(false), m_checkForGrab(false)
{
}
```

Use Properties, Messages to add a handler to the View for the `WM_RBUTTONDOWN` event. In the `OnRButtonDown` method, store the mouse position and clear the `m_grabbed` flag and set the `m_checkForGrab` flag. These will be checked in the `WM_MOUSEMOVE` handler and the `WM_RBUTTONUP` handler.

```
void CHelloGlobeView::OnRButtonDown(UINT nFlags, CPoint point)
{
  m_rmb = m_lastGrabPoint = point ;
  m_grabbed = false ;
  m_checkForGrab = true ;
  CView::OnRButtonDown(nFlags, point);
}
```

The `OnRButtonUp` method now needs modifying to ensure that the zoom out only occurs if no grab was active and the grab related flags should be cleared down.

```
void CHelloGlobeView::OnRButtonUp(UINT nFlags, CPoint point)
{
  if ( m_drawingSurface && !m_grabbed )
  {
    if ( m_drawingSurface->zoom( 25, false, false ) )
      InvalidateRect( 0, FALSE ) ;
  }
  m_grabbed = m_checkForGrab = false ;
  CView::OnRButtonUp(nFlags, point);
```

The next step is to create a mouse move handler and determine whether any grab is active. If so, then the new display centre needs to be calculated and passed to MapLink. As before, the `TSLDrawingSurface::pan` method will return true on success and false on failure. A successful pan should invalidate the window rectangle.

Use Properties, Messages to add a handler to the View for the `WM_MOUSEMOVE` event

```
void CHelloGlobeView::OnMouseMove(UINT nFlags, CPoint point)
{
  if (   m_checkForGrab
      && (   m_grabbed || abs( point.x - m_rmb.x ) > 3
          || abs( point.y - m_rmb.y ) > 3 ) )
  {
     // Calculate offset between the last point and the new point
     long du_offset_x = m_lastGrabPoint.x - point.x ;
     long du_offset_y = point.y - m_lastGrabPoint.y ;

     // Indicate a grab and remember the last grabbed point
     m_lastGrabPoint = point ;
     m_grabbed = true ;

     // Convert the offset from the last point into user units
     // No DUPerUU() function so we will take the long way round.
     double uu_per_du =   m_drawingSurface->TMCperDU()
                        / m_drawingSurface->TMCperUU() ;

     double uu_offset_x = du_offset_x * uu_per_du ;
     double uu_offset_y = du_offset_y * uu_per_du ;

     // Get the current centre point of the Drawing Surface
     double x, y, x1, y1, x2, y2 ;

     m_drawingSurface->getUUExtent( &x1, &y1, &x2, &y2 ) ;
     x = ( x1 + x2 ) / 2 ;
     y = ( y1 + y2 ) / 2 ;

     // Adjust the centre for the calculated offset
     x += uu_offset_x ;
     y += uu_offset_y ;

     // Pan and redraw the Drawing Surface
     // Request redraw only if pan is successful
     if ( m_drawingSurface->pan( x, y, false ) )
       InvalidateRect( 0, FALSE ) ;
   }
   CView::OnMouseMove(nFlags, point);
 }
```

9.2 Mouse Wheel

Many modern PC systems will have a wheel mouse available, where a wheel has replaced the middle button. This can spin and also be pressed to act as a button.

9.2.1 Wheel Support Issues

There are many issues surrounding wheel mouse support, particularly since there are many different drivers. Some drivers provide special functionality that overrides any application specific handling and replaces it with generic scrolling support. This is commonly termed 'Universal Scrolling' or 'IntelliPoint Wheel Support'.

Many MapLink applications, such as the MapLink Viewer and MapLink Studio provide specialist wheel handling to give control over the zoom factor. You can easily see if your mouse driver is overriding the application wheel support by trying the wheel in the MapLink Viewer. If this scrolls vertically instead of zooming in and out then your driver is ignoring any application specific behaviour. Some drivers can only turn this override off globally, whilst others allow it to be disabled for individual programs.

To control the generic scrolling, please review your mouse driver help, but you could try the following:

- Invoke the Control Panel, Mouse dialog.
- Select the Wheel tab.
- With some drivers, there may be a 'Universal Scrolling' check box here. Uncheck this box to remove the generic support and thereby enable application specific behaviour.
- Other drivers may have an 'Advanced' button to press. Be careful here since some drivers have two buttons labelled 'Advanced' on the same panel! Try both.
- The Advanced Wheel Support dialog may allow you to disable IntelliPoint wheel support or Universal Scrolling. You may be able to choose a specific program executable should you wish to maintain the generic support for non-wheel enabled programs.

If you have a mouse wheel but no access to the controls described above then you may need to install a new mouse driver. Your mouse driver may also have specific configuration control over what happens when you press the wheel. Again, please see your mouse driver help for further information.

9.2.2 Wheel Controlled Zoom and Pan

Assuming you have disabled generic wheel support for your application, you can add in support for zoom and pan operations using the mouse wheel. Firstly we will add wheel zoom support.

Use Properties, Messages to add a handler to the View for the `WM_MOUSEWHEEL` message. In the `OnMouseWheel` method, it either zooms in or out dependent upon the wheel direction.

```
BOOL CHelloGlobeView::OnMouseWheel(UINT nFlags,short zDelta,CPoint
pt)
{
   // Zoom in or out depending upon the wheel direction
   if (   m_drawingSurface
       && m_drawingSurface->zoom( 30.0, zDelta > 0, false ) )
   {
     InvalidateRect( NULL, FALSE );
   }
   return CView::OnMouseWheel(nFlags, zDelta, pt);
 }
```

Use Properties, Messages to add a handler to the View for the `WM_MBUTTONUP` event.

```
void CHelloGlobeView::OnMButtonUp(UINT nFlags, CPoint point)
{
  double x, y ;
  if (   m_drawingSurface
      && m_drawingSurface->DUToUU( point.x, point.y, &x, &y ) )
  {
    if ( m_drawingSurface->pan( x, y, false ) )
      InvalidateRect( 0, FALSE ) ;
  }
  CView::OnMButtonUp(nFlags, point);
}
```

10 GEOMETRY AND OVERLAYS

10.1 Entities

MapLink is often used to display vector data, either in maps or overlays. The underlying geometry model of this vector data is shown below. This model can be directly mapped onto the OpenGIS simple feature model (ignoring the User Geometry Entity). Each piece of geometry stores its coordinates in internal TMC units.

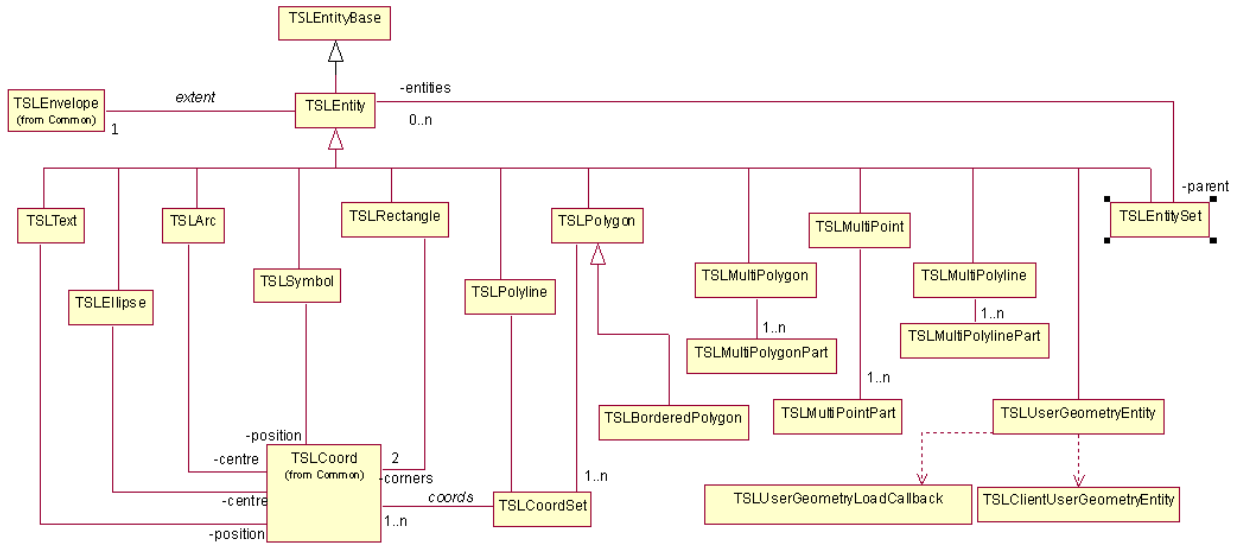


Figure 1 Geometry Hierarchy

The MapLink concept for an instantiated piece of geometry is an Entity. In the 2D SDK, these are accessed through classes derived from `TSLEntity`. Each different type of geometry has its own class. The `TSLEntityBase` class provides a common point of derivation for both 2D and 3D geometry.

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 10.6.

Several primitives define angles for rotation or reference points. These are measured with 0 degrees as the x-axis and positive anti-clockwise.

10.1.1 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.

10.1.2 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.

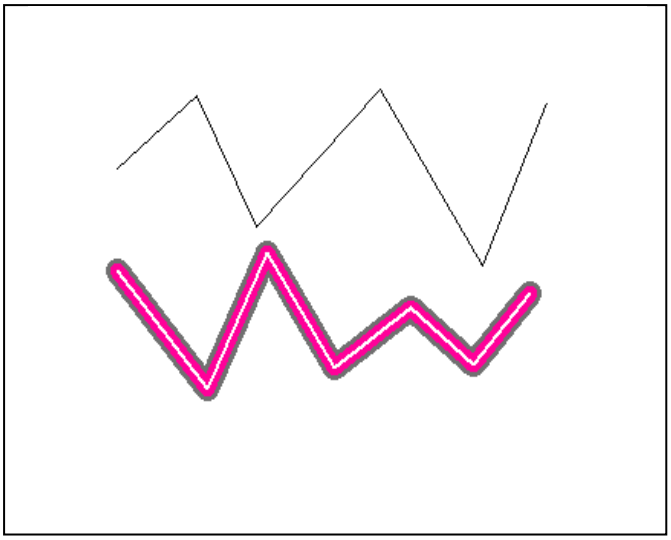


Figure 2 Polyline

10.1.3 TSLPolygon

A polygon is a two-dimensional surface. It therefore has an area and a perimeter. The rendering of a polygon may 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. MapLink 4.7 and later have additional functionality that removes single-point spikes.

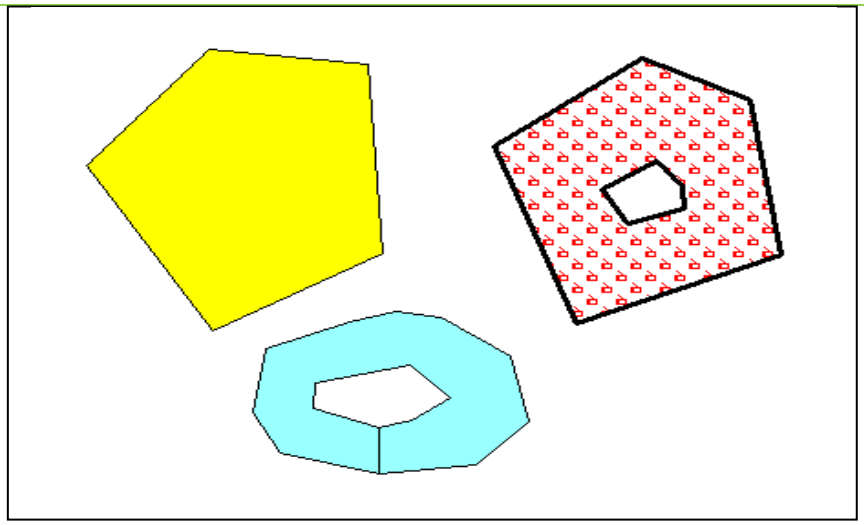


Figure 3 Polygon

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. This allows for some significant optimisations to be done through key-holing polygons so that they have only an outer ring. This gives increased performance on some platforms.

10.1.4 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.

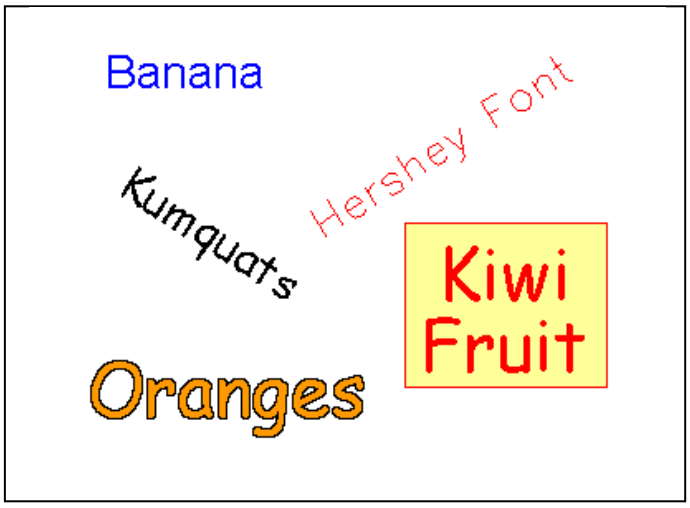


Figure 4 Text

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.

A single text object may be split over several lines, by including a carriage return (C++ `'\n'`) character amongst the text. Any alignment and background will take all lines into account.

10.1.5 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.

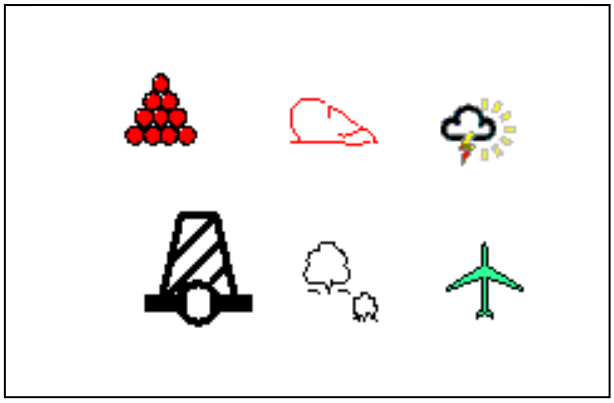


Figure 5 Symbols

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. They can be created using the Symbol Studio editor available in the MapLink bin directory. Since MapLink 4.5, vector symbols can 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. Note that not all types of drawing surface support scaling of raster symbols or images with an alpha channel.

On Windows, a special subtype of raster symbols that use Windows Icons are available. Icon symbols are displayed as fixed size and may contain an embedded transparency mask so need not be rectangular. Earlier releases can only display standard 32x32 pixel icons - Windows will automatically scale other sizes to fit into 32x32.

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.

10.1.6 Text Replacement

To use the dynamic text features, use Symbol Studio to create a symbol which contains text. If the text is prefixed by a double underscore, this indicates that the text may be dynamic.

The following dynamic text strings are recognised:

- __name

A text string of "__name" will be replaced by the name property of the symbol instance. This is the most efficient way to display a simple textual property within the symbol - e.g. a road name.
At runtime this can be replaced using `TSLEntity::name("text")`.

- __entityid

A text string of "__entityid" will be replaced by the numeric Entity ID of the symbol instance. This is the most efficient way to display a simple integer number.
At runtime this can be replaced using `TSLEntity::id(number)`. This may also be set when the entity is created.

- __ID

The text string "__ID" (where ID is any two-character string) will look up the value of the data attribute "ID" from the symbol instance. This allows any embedded data attribute to be displayed. If there is no data attribute found, then the text is not displayed.
This is the two-character ID of an attribute as setup on the `TSLDataSet` (see `TSLEntityBase::addDataSet`).

Following the '__' a valid format string may be added. This will override the defaults as defined below:

__name : %s
__entityid : %I64d
__ID : the default will depend on the mapping.

The name and/or entity ID are sometimes placed on the symbol instance by the MapLink filter. Some filters, such as the ShapeFile and MIF filters, allow you to specify that the name should be populated from attribute in the associated DBF or MID file.

10.1.7 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.

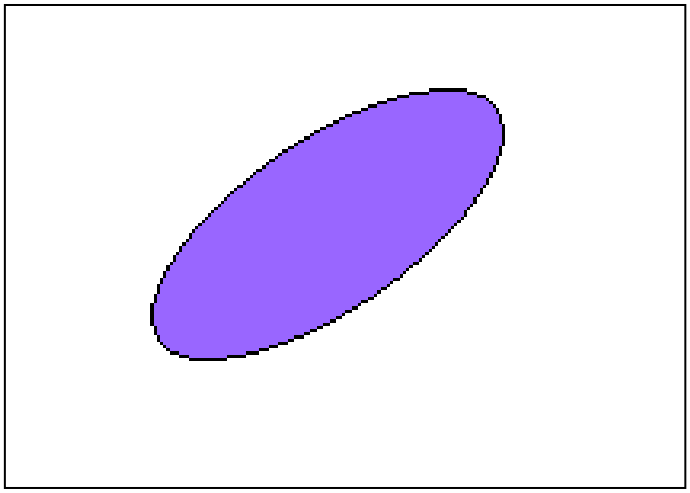


Figure 6 Ellipse

10.1.8 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.

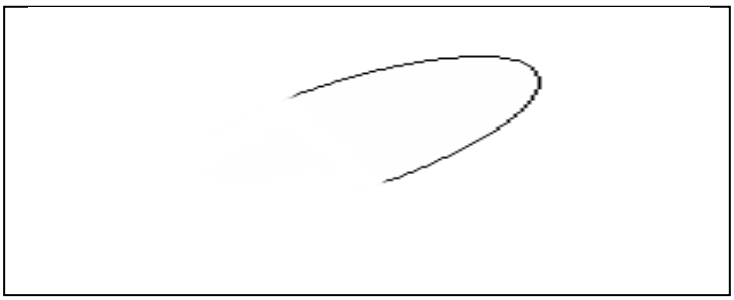


Figure 7 Arc

10.1.9 TSLRectangle

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

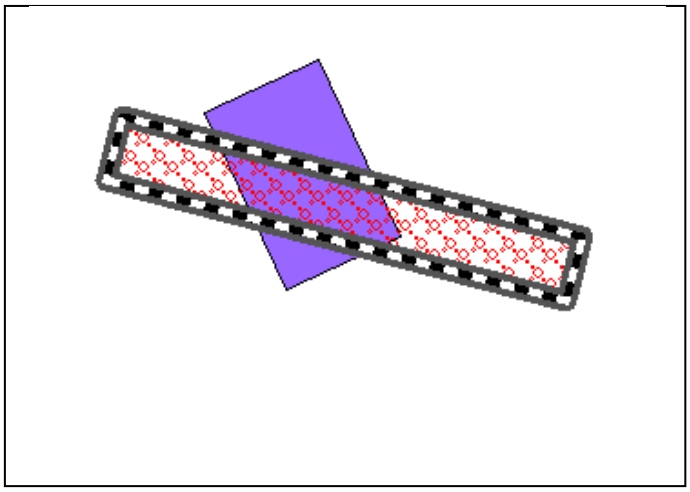


Figure 8 Rectangle

10.1.10 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.

10.1.11 TSLBorderedPolygon

This is a specialised primitive, often used in Land Registration applications. It is essentially a normal Polygon, but each edge, including those around any holes, has a separate thick border polygon associated with it. This border polygon can be drawn internally or externally to the polygon. Where these border polygons meet, MapLink performs processing to ensure that the join looks aesthetically pleasing.

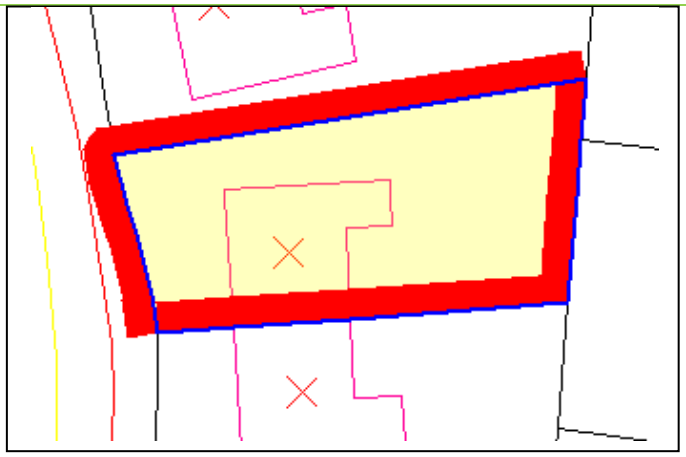


Figure 9 Bordered Polygon

10.1.12 Geodetic Primitives

A geodetic primitive is a primitive whose shape is defined by the projection of the map upon which it is drawn.

Geodetic primitives will be re-drawn if the map projection or map is changed maintaining the positions of the control points in latitude and longitude but changing shape to match the projection.

The shape of a geodetic primitive is defined by interpolating points along a geodesic path, for example; consider the path an aeroplane flies between London and Beijing. Aircraft take the shortest path, the geodesic, between points. In flat space, geodesics are straight lines; on the surface of a sphere, geodesics are the minor arcs of great circles; on the surface of the earth, approximated as an ellipsoid, geodesics are given by Vincenty's formulae.

The six geodetic primitives currently supported are shown below.

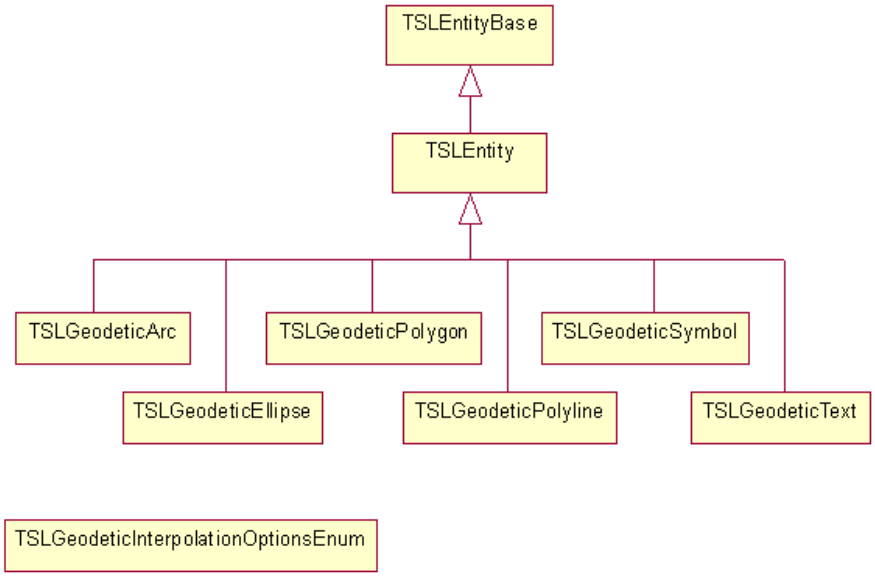


Figure 10 Geodetic Entities Hierarchy

The control points are specified as TMC values, these are then converted to latitude and longitude internally when drawn.

Values of the enumeration `TSLGeodeticInterpolationOptionsEnum` can be passed to the `interpolationOptions` method of geodetic polylines, polygons, ellipses and arcs to specify whether to use Vincenty (default) or great circles (`TSLGeodeticInterpolationOptionsGreatCircle`) to interpolate. For geodetic polylines and polygons, `interpolationOptions` can also be used to specify whether the earth should be treated as a spheroid (default) or a sphere (with `TSLGeodeticInterpolationOptionsSpherical`). Multiple flags should be combined with the bitwise OR operator.

10.1.13 TSLGeodeticPolyline

A geodetic polyline is a one-dimensional curve and defined by a sequence of at least two points.

Geodetic polylines optionally support interpolation. When this is turned off, geodetic polylines behave like normal polylines, except for changes in coordinate system. When interpolation is turned on, the lines drawn between control points are interpolated to follow geodesics along the earth's surface.

If an interpolated geodetic polyline crosses over the dateline, it will be rendered as separate pieces.

Interpolation can be turned on and off with the `interpolation` method. The post distance used for interpolating, in km, can be set and retrieved with `interpolationDistance`, and the interpolation method can be set with `interpolationOptions`.

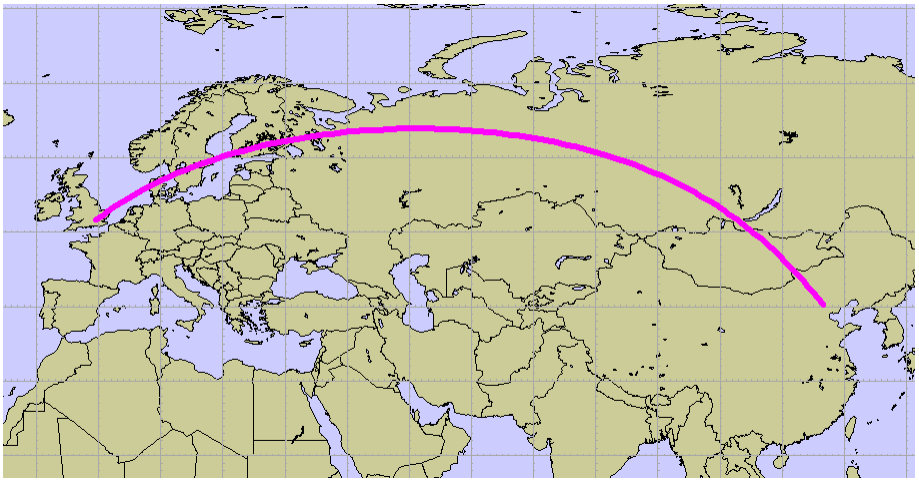


Figure 11 Two-point geodetic polyline, showing the geodesic path from Heathrow to Beijing. A Dynamic Arc map

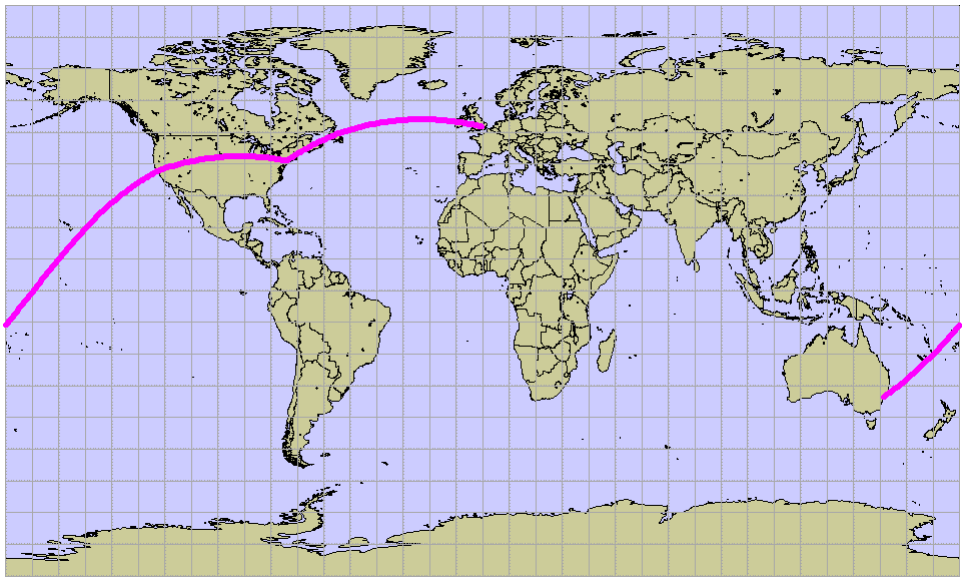


Figure 12 Single geodetic polyline with four points, travelling through Sydney, San Francisco, New York and London.

Geodetic polylines are created in a very similar way to standard polylines:

```
TSLStandardDataLayer* stdLayer = ...;
TSLDataLayer* mapLayer = ...;

TSLCoordSet* cs = new TSLCoordSet;
TSLTMC x, y;
if ( !mapLayer->latLongToTMC(51.4775, -0.461389, &x, &y) )
  ... // handle error
cs->add(x, y);
if ( !mapLayer->latLongToTMC(40.08, 116.584444, &x, &y) )
  ... // handle error
cs->add(x, y);
TSLGeodeticPolyline* polyline = stdLayer->entitySet()->
                            createGeodeticPolyline( 0, cs, true ) ;
if ( !polyline )
  ... // handle error
polyline->setRendering( TSLRenderingAttributeEdgeStyle, 1 ) ;
polyline->setRendering( TSLRenderingAttributeEdgeColour,
                        TSLComposeRGB(255,0,255) ) ;
polyline->setRendering( TSLRenderingAttributeEdgeThickness, 6 ) ;
```

Geodetic polylines can also be created directly with `TSLGeodeticPolyline::create`.

10.1.14 TSLGeodeticPolygon

A geodetic polygon is a closed shape and has a perimeter and an area.

Geodetic polygons optionally support interpolation. When this is turned off, geodetic polygons behave like normal polygons, except for changes in coordinate system. When interpolation is turned on, the lines drawn between the control points of the outer are interpolated to follow geodesics along the earth's surface.

Inners (holes) are not supported.

If an interpolated geodetic polygon crosses over the dateline, it will be rendered as separate pieces. Polygons containing any poles may not be drawn as expected.

Interpolation can be turned on and off with the `interpolation` method. The post distance used for interpolating, in km, can be set and retrieved with `interpolationDistance`, and the interpolation method can be set with `interpolationOptions`.

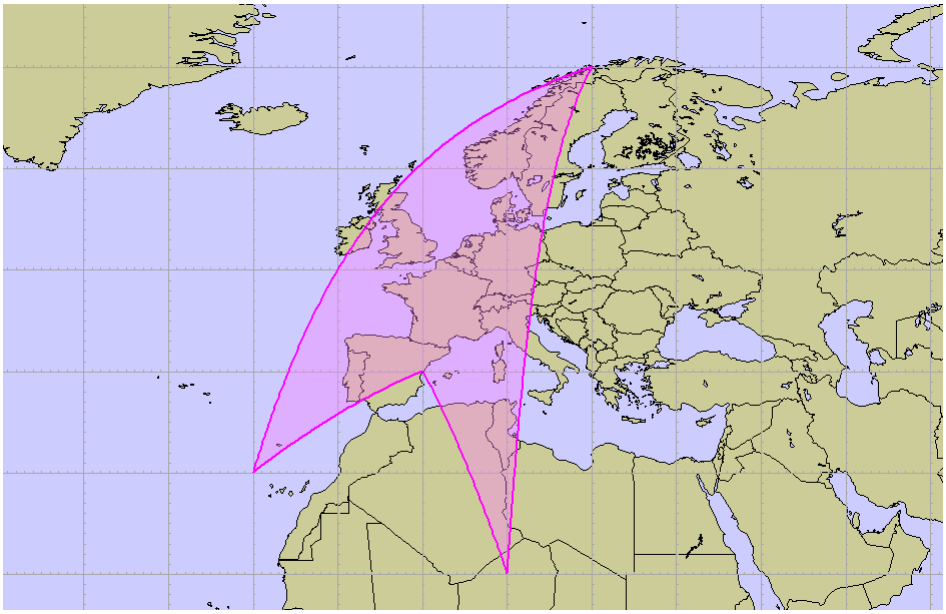


Figure 13 Four-point geodetic polygon

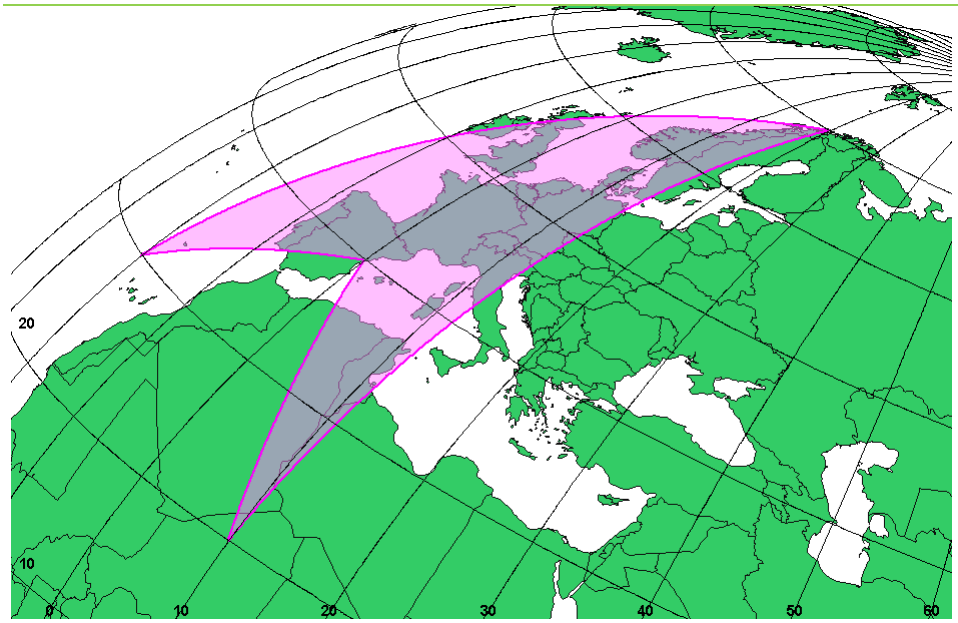


Figure 14 Four-point geodetic polygon reprojected into an orthogonal projection

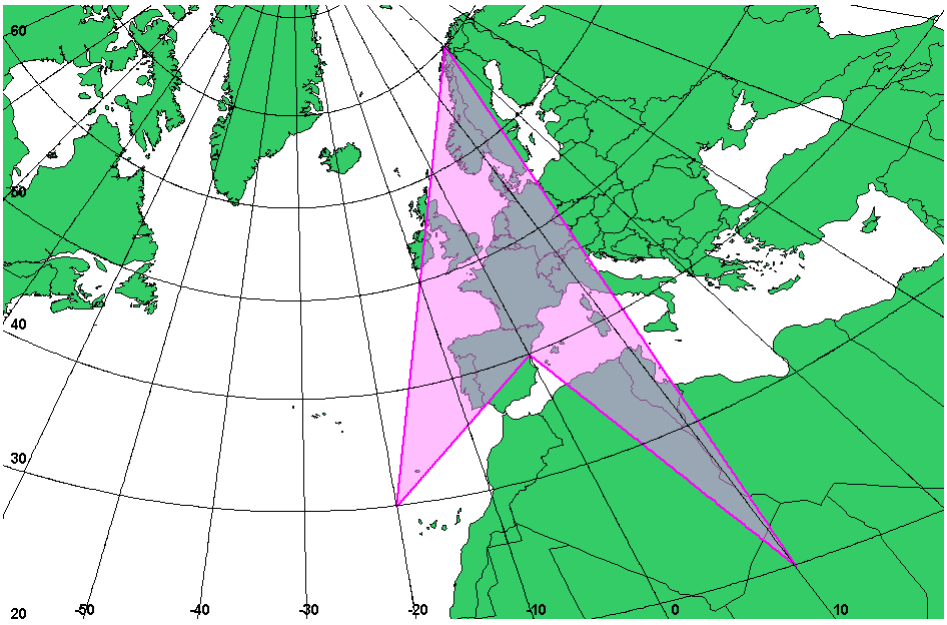


Figure 15 Four-point geodetic polygon, but on a gnomonic projection. In this projection, geodesics are straight lines, so the geodetic polygon looks like a standard polygon. The distortion in its shape is due to the centre of projection being off to one side of the geometry

Geodetic polygons are created in a very similar way to standard polygons:

```
TSLStandardDataLayer* stdLayer = ...;
TSLDataLayer* mapLayer = ...;

TSLCoordSet* cs = new TSLCoordSet;
TSLTMC x, y;
if ( !mapLayer->latLongToTMC(20.0, 10.0, &x, &y) )
  ... // handle error
cs->add(x, y);
if ( !mapLayer->latLongToTMC(40.0, 0.0, &x, &y) )
  ... // handle error
cs->add(x, y);
if ( !mapLayer->latLongToTMC(30.0, -20.0, &x, &y) )
  ... // handle error
cs->add(x, y);
if ( !mapLayer->latLongToTMC(70.0, 20.0, &x, &y) )
  ... // handle error
cs->add(x, y);
TSLGeodeticPolygon* polygon = stdLayer->entitySet()->
                              createGeodeticPolygon( 0, cs, true ) ;
if (!polygon)
  ... // handle error
polygon->setRendering( TSLRenderingAttributeEdgeStyle, 1 ) ;
polygon->setRendering( TSLRenderingAttributeEdgeColour,
                       TSLComposeRGB(255,0,255) ) ;
polygon->setRendering( TSLRenderingAttributeEdgeThickness, 2 ) ;
polygon->setRendering( TSLRenderingAttributeFillStyle, 502 ) ;
polygon->setRendering( TSLRenderingAttributeFillColour,
                       TSLComposeRGB(255,128,255) ) ;]
```

Geodetic polygons can also be created directly with `TSLGeodeticPolygon::create`.

10.1.15 TSLGeodeticEllipse

A `TSLGeodeticEllipse` primitive is a two-dimensional surface defined geometrically on the earth's surface by the centre point, x and y radial distances (in metres, not TMCs) and rotation angle. The radial distances are those before rotation is applied.

Geodetic ellipses are created the same way as standard ellipses, except the x and y radii are floating-point numbers, representing the geodesic distance from the centre in metres. Metres are used because TMCs can distort and wrap around near the edges of maps.

Geodetic ellipses also provide control over the interpolation of their edge. The interpolation step angle, in radians, can be set using `interpolationAngleDelta`, and the interpolation method can be set with `interpolationOptions`.

If a geodetic ellipse crosses the dateline, it will be rendered as separate pieces. A geodetic ellipse can cover a pole.

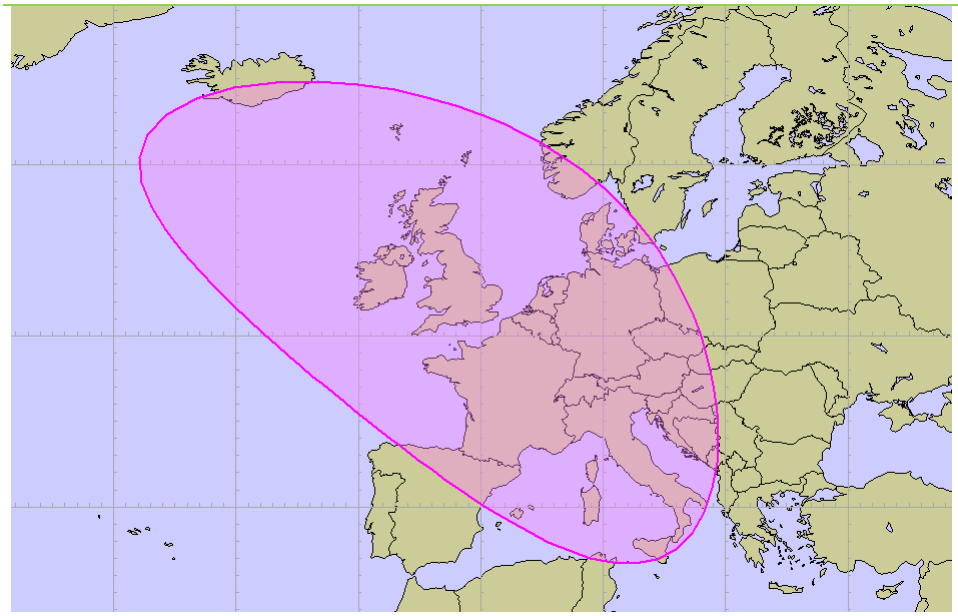


Figure 16 Geodetic ellipse centred on London; x-radius 1000km, y-radius 2000km, rotation 45°.

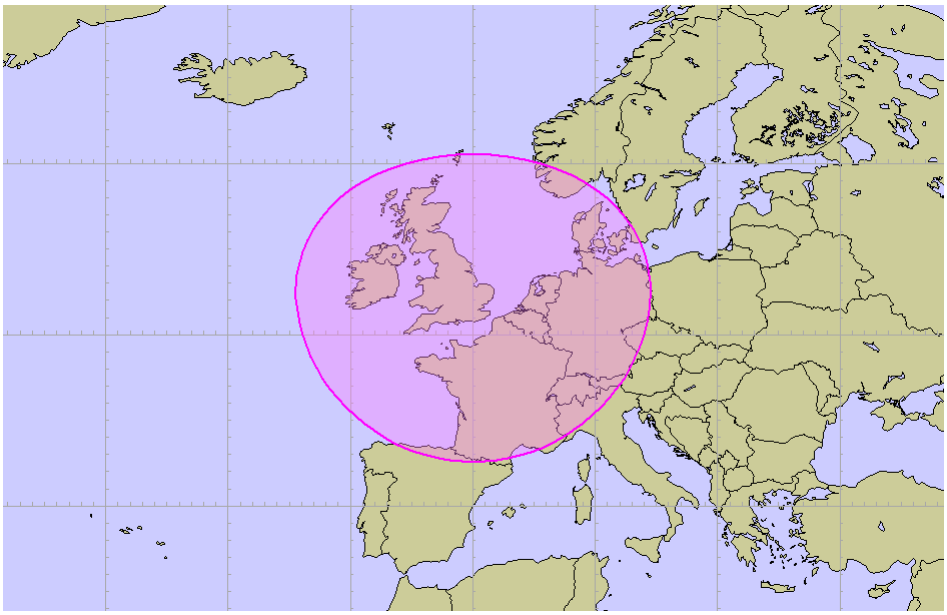


Figure 17 Geodetic ellipse centred on London; x- and y-radius 1000km

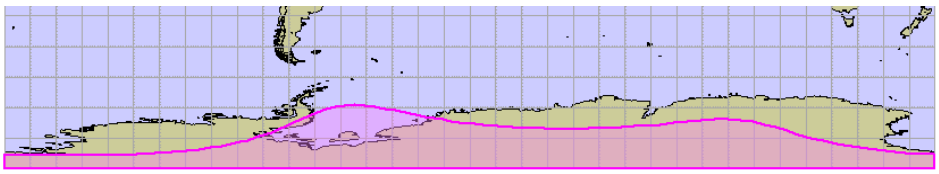


Figure 18 Geodetic ellipse centred on 85°S 0°E; x-radius 1000km, y-radius 2000km, rotation 60°.

Geodetic ellipses are created in a similar way to standard ellipses, except for the radii:

```
TSLStandardDataLayer* stdLayer = ...;
TSLDataLayer* mapLayer = ...;

TSLTMC x, y;
if ( !mapLayer->latLongToTMC(51.5, 0.05, &x, &y) )
  ... // handle error
TSLGeodeticEllipse* ellipse = stdLayer->entitySet()->
                              createGeodeticEllipse( 0, x, y,
                                1000000.0, 2000000.0, M_PI/4.0 );
if (!ellipse)
  ... // handle error
ellipse->setRendering( TSLRenderingAttributeEdgeStyle, 1 ) ;
ellipse->setRendering( TSLRenderingAttributeEdgeColour,
                       TSLComposeRGB(255,0,255) ) ;
ellipse->setRendering( TSLRenderingAttributeEdgeThickness, 2 ) ;
ellipse->setRendering( TSLRenderingAttributeFillStyle, 502 ) ;
ellipse->setRendering( TSLRenderingAttributeFillColour,
                       TSLComposeRGB(255,128,255) ) ;
```

Geodetic ellipses can also be created directly with `TSLGeodeticEllipse::create`.

10.1.16 TSLGeodeticArc

The `TSLGeodeticArc` primitive is a one-dimensional curve, which is a portion of the circumference of a geodetic ellipse. It therefore has length but no area. It is specified geometrically on the surface of the earth by the centre of the ellipse, the x and y radial distances (in metres, not TMCs) 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 geodetic ellipse to be rotated. The sweep of the geodetic arc is anti-clockwise from start angle to end angle.

Geodetic arcs are created the same way as standard arcs, except the x and y radii are floating-point numbers, representing the geodesic distance from the centre in metres.

Geodetic arcs also provide control over their interpolation. The interpolation step angle, in radians, can be set using `interpolationAngleDelta`, and the interpolation method can be set with `interpolationOptions`.

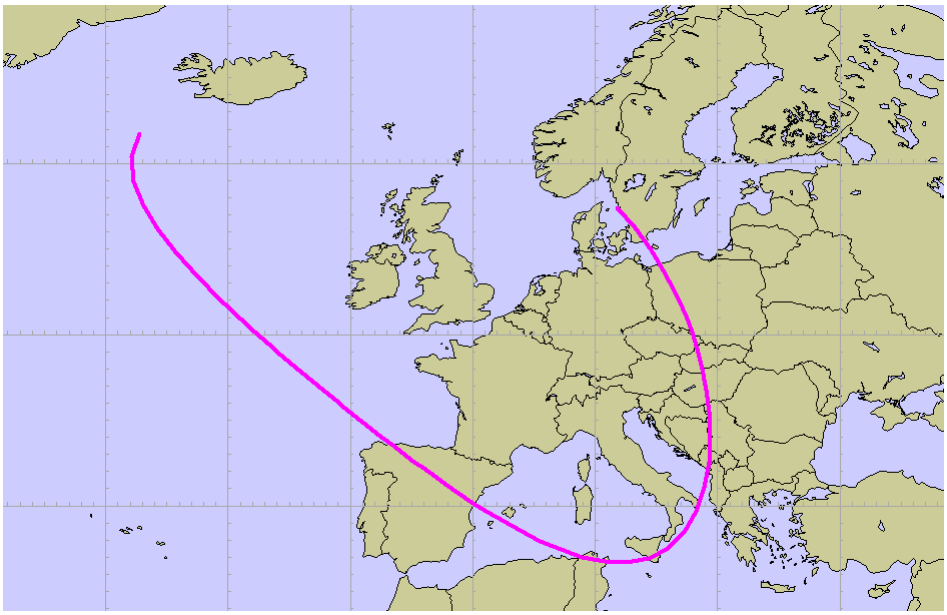


Figure 19 Geodetic arc centred on London; x-radius 1000km, y-radius 2000km, rotation 45°.

Geodetic arcs are created in a similar way to standard arcs, except for the radii:

```
TSLStandardDataLayer* stdLayer = ...;
TSLDataLayer* mapLayer = ...;

TSLTMC x, y;
if ( !mapLayer->latLongToTMC(51.5, 0.05, &x, &y) )
  ... // handle error
TSLGeodeticArc* arc = stdLayer->entitySet()->
                      createGeodeticArc( 0, M_PI/2, 2*M_PI, x, y,
                              1000000.0, 2000000.0, M_PI/4.0 );
if (!arc)
  ... // handle error
arc->setRendering( TSLRenderingAttributeEdgeStyle, 1 ) ;
arc->setRendering( TSLRenderingAttributeEdgeColour,
                   TSLComposeRGB(255,0,255) ) ;
arc->setRendering( TSLRenderingAttributeEdgeThickness, 4 ) ;
```

Geodetic arcs can also be created directly with `TSLGeodeticArc::create`.

10.1.17 TSLGeodeticText

A `TSLGeodeticText` object consists of a single position coordinate and a text string, and behaves identically to a `TSLText`, except for coordinate system changes.

10.1.18 TSLGeodeticSymbol

A `TSLGeodeticSymbol` is specified by a single coordinate, and behaves almost identically to a `TSLSymbol`, except for coordinate system changes.

10.2 User Geometry

A user geometry entity allows the user to create custom-drawn geometry upon standard data layers. User geometry can be saved to and loaded from TMF files. A piece of 2D user geometry is composed of two parts, the entity (an instance of `TSLUserGeometryEntity`, managed by MapLink) and the client (an instance derived from `TSLClientUserGeometryEntity`, managed by the user).

10.2.1 TSLUserGeometryEntity

Instances of `TSLUserGeometryEntity` can be added to standard data layers, and are allocated and deallocated by MapLink. Create instances by calling `TSLUserGeometryEntity::create`, or by calling `createUserGeometry` on a `TSLEntitySet`. The client of a user geometry entity can be set and retrieved by calling `setClientUserGeometryEntity` and `getClientUserGeometryEntity`, respectively.

`create`, `createUserGeometry`, `setClientUserGeometryEntity` and load callback functions (see section 10.2.3) all provide an `ownsClient` flag. If true, then MapLink will automatically `delete` the client if it is replaced with `setClientUserGeometryEntity` or when the entity is destroyed. If false, the user will have to destroy the client. This must be false if the user's code is compiled with a different compiler or runtime library version to MapLink.

Creating and destroying user geometry:

```
TSLStandardDataLayer* stdLayer = ...;
TSLClientUserGeometryEntity* client = new ...;

TSLUserGeometryEntity* entity = stdLayer->entitySet()->
                                        createUserGeometry(client, false);
if (!entity)
  ... // handle error

...

entity->destroy();
delete client;   // don't need this if ownsClient is true
```

10.2.2 TSLClientUserGeometryEntity

The user creates clients by deriving from `TSLClientUserGeometryEntity` and creating their own instances of these subclasses. A client can then be attached to an entity as explained above.

At a minimum, the user must override the virtual `draw` method.

Here is an example partial implementation of a user geometry client:

```
class RectangleClient : public TSLClientUserGeometryEntity
{
private:
  TSLEnvelope m_extent;

public:
  // Constructor
  RectangleClient(TSLTMC left, TSLTMC bottom, TSLTMC right, TSLTMC top)
  {
    m_extent.corners( left, bottom, right, top );  // bounding box
  }

  // Destructor
  virtual ~RectangleClient()
  {
  }

  // Draw a rectangle using rendering interface
  virtual bool draw(int uniqueSurfaceID,
                    TSLRenderingInterface* renderingInterface,
                    const TSLEnvelope& extent, TSLRenderLevel renderLevel,
                    double screenResolution)
  {
    const int blue = TSLDrawingSurface::getIDOfNearestColour(0,0,255);

    // Construct a rectangle
    TSLCoord coords[4];   // bottom left, top left, top right, bottom right
    coords[0] = m_extent.bottomLeft();
    coords[1] = TSLCoord( m_extent.bottomLeft().x(), m_extent.topRight().y() );
    coords[2] = m_extent.topRight();
    coords[3] = TSLCoord( m_extent.topRight().x(), m_extent.bottomLeft().y() );

    // Set up rendering attributes - translucent rectangle
    renderingInterface->setupEdgeAttributes( -1, 0, 0.0 );
    renderingInterface->setupAreaAttributes( 503, blue );

    // Attempt to draw rectangle, return false if fails
    return renderingInterface->drawPolygon(coords, _countof(coords));
  }

  // Save the rectangle
  virtual int save(TSLofstream& stream)
  {
    // Stream out data
    ...
    return RECTANGLE_USER_GEOMETRY_ID;  // unique ID of the user geometry type
  }

  // Return the envelope for this rectangle
  virtual TSLEnvelope envelope(int uniqueSurfaceID)
  {
    return m_extent;
  }
};
```

10.2.3 Loading and saving user geometry

If the user wants their user geometry classes to be saved and loaded along with other types of geometry, they need to override the `save` method on the client, and to provide a load callback function to the static method `TSLUserGeometryEntity::registerUserGeometryClientLoadCallback`.

The `save` method on the client should return a positive integer to identify the type of user geometry. These numbers should be unique as they can be passed to any registered load callback function. It is suggested that the developer publish and track these identifiers.

It is also suggested that the developer saves, along with any geometry data, a company identifier, a byte-order mark, a geometry type ID and a version number.

To register a load callback function, a pointer to it must be passed to `TSLUserGeometryEntity::registerUserGeometryClientLoadCallback`. The pointer should have type `TSLUserGeometryLoadCallback` (which is a function pointer typedef). The pointer will be added to a list; when user geometry is loaded, each function on the list will be called until one returns non-NULL.

Setting a load callback function:

```
TSLUserGeometryEntity::
          registerUserGeometryClientLoadCallback(loadUserGeometryCallback);
```

Here is a skeleton load callback function:

```
static TSLClientUserGeometryEntity* loadUserGeometryCallback(
                                          TSLifstream& stream,
                                          int userGeometryID,
                                          bool& assumeOwnership)
{
  // whether returned entities will be freed by MapLink:
  assumeOwnership = ...;

  switch (userGeometryID)
  {
    case RECTANGLE_USER_GEOMETRY_ID:
      ... // stream in client and return it

    ... // etc

    default:
      return NULL;
  }
```

10.3 Data Layers

The usual method of displaying fairly static data in MapLink is via the `TSLStandardDataLayer`. This allows any of the geometric Entities to be displayed and overlaid on top of a map. Of course, you can have any number of Data Layers displayed on a single Drawing Surface, so you are not unduly restricted.

Unlike a `TSLMapDataLayer`, the `TSLStandardDataLayer` has no coordinate system of its own. Instead, it uses the coordinate system of the Drawing Surfaces to which it is attached. You should therefore be certain that if the `TSLStandardDataLayer` is displayed on multiple Drawing Surfaces then they have consistent coordinate systems.

The `TSLDrawingSurface` assumes the coordinate system of the last `TSLMapDataLayer` instance added to the surface. This is not necessary the topmost map data layer since the order can be modified by the application.

Each `TSLStandardDataLayer` contains a `TSLEntitySet` that is used as a container for the Data Layers Entities. To add Entities to the Data Layer, simply query the Data Layer for its `TSLEntitySet` and then use the `createXxxx` methods of the Entity Set. The Entities created will be added to Data Layer and displayed on the next redraw. If you are using double buffering, you should call `notifyChanged` on the Data Layer to indicate that the contents have changed.

The `TSLStandardDataLayer` has several storage methods available, for loading and saving its contents via a file or buffer. The buffer or file that is created by this process is a proprietary binary format that may be written to a database blob if required.

It is also possible to load and save the rendering and feature list configuration of the `TSLStandardDataLayer`. These are the `loadDataWithConfig` and `saveDataWithConfig` methods.

10.3.1 Utility Classes used during Entity Creation

There are several classes needed when creating Entities. These are various ways of defining positions and sizes. In general, since the Entities are defined independent of Drawing Surfaces, they use internal TMC coordinate space rather than any particular map coordinate system. If necessary, the coordinate system conversion methods available on Drawing Surfaces and map Data Layers could be used to transform positions into the TMC coordinate space.

A single point is usually specified either by passing individual x, y parameters or by using the `TSLCoord` utility class. Where an indeterminate number of points must be specified, for example to create a polygon, then an instance of a `TSLCoordSet` class is most often used. Another commonly used class is `TSLEnvelope`. This holds two coordinates forming a rectangle and is most often used to pass extents and areas. Full details of the methods available on these classes may be found in the detailed online SDK documentation.

10.4 GARS, MGRS and Latitude/Longitude data layers

There are three data layers within MapLink to handle Latitude/Longitude called `TSLLatLongGridDataLayer`, GARS Grid called `TSLGARSDataLayer` and Military Grid Reference System (MGRS) and/or the Universal Transverse Mercator (UTM) grid called `TSLMGRSGridDataLayer`.

10.4.1 The TSLMGRSGridDataLayer

The `TSLMGRSGridDataLayer` works in two modes, an "automatic" mode and a "single zone" mode. The automatic mode is designed to give reasonable grid lines in all projections, coordinate systems and at any resolution; However, being a general solution it may not be exactly what the user wants. Some projections have very curved grid lines away from the centre of the projection, and there may be errors towards the edge of the map.

The single zone mode is designed to work with transverse Mercator maps in the zone and band specified, although it also will work in other projections.

In general, if you know the MGRS grid zone and band, then the single zone will be more appropriate. For whole world maps or those displayed before the user selects a zone and band, it is best to use the automatic mode. It is possible to turn off the single zone mode by giving it a zone of -1.

The `TSLMGRSGridDataLayer` may display MGRS, UTM and latitude/longitude grids. Which grids are displayed is controlled by the line and text attributes given below.

The MGRS grid has special features to deal with Scandinavia and is widened to 12 degrees between 72 and 84 degrees north. The latitude / longitude grid incorporates these special features as does the MGRS grid when displayed.

It is also possible to customise the grids displayed, for which grid lines are displayed, the line styles, whether grid squares are named, whether grid lines are labelled and the styles of the text displayed.

For the `TSLMGRSGridDataLayer` the following lines are configurable:

- `"lon6Degree"`
- `"lonDegree"`
- `"lonMinute"`
- `"lat8Degree"`
- `"latDegree"`
- `"latMinute"`
- `"utm1km"`
- `"utm10km"`
- `"utm100km"`

And the following text labels:

- `"utmLabel"`

- "mgrsLabel",
- "gridLineLabel"
- "degreeLineLabel"

For a grid line to be displayed it must have a colour greater than zero and a thickness greater than zero. For a text label to be displayed, it must have a colour greater than zero.

To configure which grids are displayed:

An MGRS grid is displayed when the "utm1km", "utm10km" or "utm100km" grid lines are displayed and the "mgrsLabel" is displayed.
A UTM grid is displayed if the MGRS grid is not displayed and the "utm1km", "utm10km" or "utm100km" grid lines are displayed and the "utmLabel" is displayed.
A lat/long grid is displayed when any of the "lon6Degree", "lonDegree", "lonMinute", "lat8Degree", "latDegree" or "latMinute" are displayed irrespective of whether either or both the MRGS and UTM grids are displayed.

To create an MGRS Grid Data Layer:

```
m_mgrsGridLayer = new TSLMGRSGridDataLayer ;
m_mgrsGridLayer->setMapDataLayer(m_mapDataLayer);
```

To initialise the MGRS grid line styles:

```
long utm1kmC, utm1kmS, utm1kmT ;
long utm10kmC, utm10kmS, utm10kmT ;
long utm100kmC, utm100kmS, utm100kmT ;

TSLProfileHelper::lookupProfile("gridUtm1kmColour",&utm1kmC,
                                        getIDOfNearestColour( "0,0,127" ) );
TSLProfileHelper::lookupProfile("gridUtm10kmColour",&utm10kmC,
       getIDOfNearestColour( "0,0,192" ) );
TSLProfileHelper::lookupProfile("gridUtm100kmColour",&utm100kmC,
       getIDOfNearestColour( "0,0,255" ) );
TSLProfileHelper::lookupProfile("gridUtm1kmStyle",&utm1kmS, 6 );
TSLProfileHelper::lookupProfile("gridUtm10kmStyle",&utm10kmS, 3 );
TSLProfileHelper::lookupProfile("gridUtm100kmStyle",&utm100kmS, 1 );
TSLProfileHelper::lookupProfile("gridUtm1kmThickness",&utm1kmT, 1 );
TSLProfileHelper::lookupProfile("gridUtm10kmThickness",&utm10kmT, 1 );
TSLProfileHelper::lookupProfile("gridUtm100kmThickness",&utm100kmT, 2);

m_mgrsGridLayer->setFeatureRendering("utm1km", 0, TSLRenderingAttributeEdgeColour,
                                     utm1kmC);
m_mgrsGridLayer->setFeatureRendering("utm1km", 0, TSLRenderingAttributeEdgeStyle,
                                     utm1kmS);
m_mgrsGridLayer->setFeatureRendering("utm1km", 0, TSLRenderingAttributeEdgeThickness,
                                    (double)utm1kmT);
m_mgrsGridLayer->setFeatureRendering("utm10km", 0, TSLRenderingAttributeEdgeColour,
                                     utm10kmC);
m_mgrsGridLayer->setFeatureRendering("utm10km", 0, TSLRenderingAttributeEdgeStyle,
                                     utm10kmS);
m_mgrsGridLayer->setFeatureRendering("utm01km", 0, TSLRenderingAttributeEdgeThickness,
                                    (double)utm10kmT);
m_mgrsGridLayer->setFeatureRendering("utm100km", 0, TSLRenderingAttributeEdgeColour,
                                    utm100kmC);
m_mgrsGridLayer->setFeatureRendering("utm100km", 0, TSLRenderingAttributeEdgeStyle,
                                    utm100kmS);
m_mgrsGridLayer->setFeatureRendering("utm100km", 0, TSLRenderingAttributeEdgeThickness,
                                     (double)utm100kmT);
```

To initialise the MGRS grid text styles:

```
initialiseLabel(m_mgrsGridLayer, "mgrsLabel );
initialiseLabel(m_mgrsGridLayer, "gridLineLabel );
```

Where the method `initialiseLabel` sets the following attributes for the text:

```
TSLRenderingAttributeRenderLevel
TSLRenderingAttributeTextFont
TSLRenderingAttributeTextColour
TSLRenderingAttributeTextSizeFactor
TSLRenderingAttributeTextSizeFactorUnits
TSLRenderingAttributeTextHorizontalAlignment
TSLRenderingAttributeTextVerticalAlignment
TSLRenderingAttributeTextBackgroundMode
TSLRenderingAttributeTextBackgroundColour
TSLRenderingAttributeTextBackgroundEdgeColour
TSLRenderingAttributeTextBackgroundStyle
TSLRenderingAttributeTextOffsetUnits
TSLRenderingAttributeTextRotatable
TSLRenderingAttributeTextMinPixelHeight
TSLRenderingAttributeTextMaxPixelHeight
```

Similarly, the UTM grid lines and labels may be initialised if they are required to be displayed. If the lat/long grid lines and labels are required, they should be initialised too.

Add the `TSLMGRSGridDataLayer` to the surface and set its visibility.

```
surface->addDataLayer( m_mgrsGridLayer, m_mgrsGridLayerName ) ;
surface->setDataLayerProps( m_mgrsGridLayerName, TSLPropertyVisible,
                                    m_mgrsGridLayerVisible ) ;
```

To set the "single zone" mode:

```
m_mgrsGridLayer->setZone(zone, band);
```

and back to "automatic" mode:

```
m_mgrsGridLayer->setZone(-1, 0);
```

10.4.2 The TSLLatLongGridDataLayer

The `TSLLatLongGridDataLayer` displays a latitude / longitude grid like that which may be displayed by the `TSLMGRSGridDataLayer`. However, there are a few differences:

- The major grid lines are on 6-degree boundaries for both latitude and longitude.
- Ticks on the grid lines show subdivisions.
- The special MGRS grid features dealing with Scandinavia are not included.
- There are different feature classes used. See the class documentation for further details.

To create the `TSLLatLongGridDataLayer`:

```
m_latLonGridLayer = new TSLLatLongGridDataLayer ;
m_latLonGridLayer->setMapDataLayer(m_mapDataLayer);
```

To enable ticks on the grid lines showing divisions:

```
m_latLonGridLayer->ticks(true);
```

10.4.3 The TSLGARSGridDataLayer

This draws a GARS (Global Area Reference System) grid.

More information about the GARs Grid can be found here:

http://earth-info.nga.mil/GandG/coordsys/grids/gars.html.

10.5 Additional Data Layers

10.5.1 Custom Data Layer

The Custom Data Layer concept allows Developers to draw complex content themselves. A Custom Data Layer has the following features:

- The layer can be a coordinate providing layer, for example a `TSLMapDataLayer` is a coordinate providing layer.
- Support for editing of MapLink geometry.
- Ability to notify the Drawing Surface of the ideal Active layer (required for 3D and Accelerator SDK).
- Ability to contain other MapLink 2D layers and draw them when required by the application. This is a concept similar to a `TSLMapDataLayer` which can contain multiple layers (different resolutions).
- Access to the screen resolution and layer properties (`TSLPropertyEnum`).
- Ability to drawing data using Native drawing code of the `TSLRenderingInterface`.

This functionality permits users to create their own layer to support, for example, proprietary Web Map Servers (ones which do not conform to the OGC WMS Standard) or display of Vector data from a WFS server (see the "WFS Client SDK").

The MapLink Pro team has extensive experience creating specific visualisation layers. If you require a project specific layer then please contact Sales to discuss the possibility for consultancy to help implement a layer.

10.5.2 Standard Data Layer

This is a standard component.

This layer is for user created geometry overlays. The layer is covered in section 11.

10.5.3 Dynamic Data Object Layer

This is a standard component.

This layer is designed for displaying large numbers of tracks. The layer is covered in section 14.

10.5.4 S57/S63 Data Layer

This layer is an optional component.

The layer follows the IHO S63 specification and provides an OEM the ability to create a compliant solution for displaying S63 data.

10.5.5 CADRG Data Layer

This layer is an optional component.

The layer provides the ability to display CADRG/CIB data directly within an application using the 2D Drawing surfaces including the Accelerator surfaces and the 3D Drawing Surface.

10.5.6 WMS DataLayer

This is a standard component.

This layer allows you to display data from WMS Servers using 2D Drawing surfaces including the Accelerator surfaces and the 3D Drawing Surface. See section 12.13.

10.5.7 WMTS DataLayer

This is a standard component.

This layer allows you to display data from WMS Servers using 2D Drawing surfaces including the Accelerator surfaces.

10.5.8 KML DataLayer

This is a standard component.

This layer allows you to display KML/KMZ data using 2D Drawing surfaces including the Accelerator surfaces. Please see the sample for additional information.

10.5.9 Filter Data Layers

Filter Data Layers are essentially mini MapLink Studio layers that allow users to direct import data, re-projection and save the results. The layer only offers a subset of data processing options. See section 12.12.

If the filter you wish to use is not currently supported, please contact sales to discuss.

10.5.10 Raster Filter Data Layer

This is a standard layer.

This layer provides access to the Raster filter and the GeoTIFF filter.

10.5.11 NITF Filter Data Layer

This is an optional component.

This layer provides access to the NITF filter.

The NITF Filter Data Layer is configured in a similar way as the Raster Filter Data Layer. Please see the previous section for an example.

10.5.12 Direct Import Data layer

This is a standard component.

This layer allows users to load a wide variety of data formats at runtime in a scalable and performant manner. See section 14.

10.6 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'.

Many of the rendering attributes refer to configuration files such as '`tsllinestyles.dat`'. See section 12.6 for further details about the contents of these files.

10.6.1 Rendering Attributes

Wherever they are defined, the graphical properties are split into 5 categories and 3 types.

10.6.2 Generic Attributes

These are available on all Entities, regardless of type.

- `TSLRenderingAttributeFeatureID`: 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.
- `TSLRenderingAttributeRenderLevel`: Valid values are -5 to +5. The default is 0.
- `TSLRenderingAttributeVisible`: Boolean flag which indicates whether the Entity should be drawn. The default is true.
- `TSLRenderingAttributeSelectable`: Boolean flag that indicates whether the Entity can be found when selecting objects using the Editor SDK or when searching the data using the find and query methods of the Drawing Surface and Data Layer. Note that the Data Layer properties `TSLPropertyDetect` and `TSLPropertySelect` are also considered when searching and selecting. The default is true.
- `TSLRenderingAttributeReadOnly`: Boolean flag that indicates whether the attributes defined on an Entity are read-only. This flag can be used to inhibit modification through the Editor SDK. Of course, this attribute itself cannot be read-only otherwise it cannot be turned off! The default is false.

10.6.3 Line Rendering Attributes

These are available on one-dimensional Entities such as Polylines and Arcs. They are:

- `TSLRenderingAttributeEdgeColour`: This value must be an index from the `tslcolours.dat` file, the currently loaded map palette or a 24-bit colour (see `TSLColourHelper` API Documentation). The default is –1, which inhibits display of the Entity.
- `TSLRenderingAttributeEdgeStyle`: This value must be an index from the tsllinestyles.dat file. The default is –1, which inhibits display of the Entity.
- `TSLRenderingAttributeEdgeThicknessUnits`: 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 `TSLDimensionUnitsPixels`.
- `TSLRenderingAttributeEdgeThickness`: This value is in the units defined by the `TSLRenderingAttributeEdgeThicknessUnits` 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.

10.6.4 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:

- `TSLRenderingAttributeFillColour`: This value must be an index from the `tslcolours.dat` file, the currently loaded map palette or a 24-bit colour (see `TSLColourHelper` API Documentation). The default is –1, which inhibits display of the fill potentially leaving just the edge of the Entity.
- `TSLRenderingAttributeFillStyle`: 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.
- `TSLRenderingAttributeExteriorEdgeColour`: This value must be an index from the `tslcolours.dat` file, the currently loaded map palette or a 24 bit colour (see `TSLColourHelper` API Documentation). 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.
- `TSLRenderingAttributeExteriorEdgeStyle`: 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.

- `TSLRenderingAttributeExteriorEdgeThicknessUnits`: 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`.
- `TSLRenderingAttributeExteriorEdgeThickness`: 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.
- `TSLRenderingAttributeBorderWidth`: This value, in internal TMC units, is the width of the internal or external border of a `TSLBorderedPolygon` object. It has no effect on a normal polygon. This may be displayed in addition to the standard edge of the polygon. A value of 0 indicates that no border is displayed. Under these circumstances, a `TSLBorderedPolygon` is displayed as a normal `TSLPolygon`.
- `TSLRenderingAttributeBorderColour`: This value must be an index from the `tslcolours.dat` file, the currently loaded map palette or a 24 bit colour (see `TSLColourHelper` API Documentation). It defines the colour of the internal or external border of a `TSLBorderedPolygon`. The default is –1, which inhibits display of the border and hence it is displayed as a normal `TSLPolygon`.

10.6.5 Text Rendering Attributes

These are available on Text Entities. They are:

- `TSLRenderingAttributeTextColour`: This value must be an index from the `tslcolours.dat` file, the currently loaded map palette or a 24 bit colour (see `TSLColourHelper` API Documentation). The default is –1, which inhibits display of the Entity.
- `TSLRenderingAttributeTextFont`: This value must be an index from the tslfonts.dat file. The default is –1, which inhibits display of the text. Note that the contents of the tslfonts.dat file are operating system dependant and so may not give an exact match if displayed on different machines.
- `TSLRenderingAttributeTextSizeFactor`: 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 `TSLRenderingAttributeTextSizeFactorUnits`. The default is 0, which inhibits display of the text.

- `TSLRenderingAttributeTextSizeFactorUnits`: This value is one of `TSLDimensionUnits` enum, and determines how the `TSLRenderingAttributeTextSizeFactor` value is interpreted. Typical values allow the height of the text to be defined in points, Map Units, internal TMC units or device units. An additional value for text and symbol size factors is `TSLDimensionUnitsScaleFactor`. This makes MapLink calculate the actual size of the object by multiplying the `TSLRenderingAttributeTextSizeFactor` by the TMC height stored on the Entity. This facility is included mainly for backwards compatibility and it is recommended that new code does not use this. However, again for backwards compatibility, the default is `TSLDimensionUnitsScaleFactor`!
- `TSLRenderingAttributeTextMinPixelHeight`: 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 `TSLPropertyMinTextHeight` property.
- `TSLRenderingAttributeTextMaxPixelHeight`: 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 `TSLPropertyMaxTextHeight` property.
- `TSLRenderingAttributeTextOffsetX`: 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 `TSLRenderingAttributeTextOffsetUnits` property.
- `TSLRenderingAttributeTextOffsetY`: 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 `TSLRenderingAttributeTextOffsetUnits` property.
- `TSLRenderingAttributeTextOffsetUnits`: This value is one of `TSLDimensionUnits` enum, and determines how the `TSLRenderingAttributeTextOffsetX/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 `TSLDimensionUnitsUndefined`, which in this case is interpreted as pixels.

- `TSLRenderingAttributeTextVerticalAlignment`: 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 `TSLVerticalAlignmentUndefined`.
- `TSLRenderingAttributeTextHorizontalAlignment`: 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 `TSLHorizontalAlignmentUndefined`.

`TSLRenderingAttributeTextBackgroundMode`: 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 halo effect may be applied to either raster or Hershey vector text and will always be a single pixel in the configured text background colour. This effect renders the text multiple times, so it can have a performance hit. We recommend that you verify the performance on your target system.
The default value is `TSLTextBackgroundModeNone`.

- `TSLRenderingAttributeTextBackgroundColour`: This value must be an index from the `tslcolours.dat` file, the currently loaded map palette or a 24-bit colour (see `TSLColourHelper` API Documentation). When using rectangle backgrounds, this attribute defines the fill colour. When using halo backgrounds, this attribute defines the outline colour. The default is –1, which inhibits display of the background.
- `TSLRenderingAttributeTextBackgroundStyle`: 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.
- `TSLRenderingAttributeTextBackgroundEdgeColour`: This value must be an index from the `tslcolours.dat` file, the currently loaded map palette or

a 24-bit colour (see `TSLColourHelper` API Documentation). The default is –1, which inhibits display of any background rectangle edge.

- `TSLRenderingAttributeTextFixedHeight`: Deprecated, use `TSLRenderingAttributeTextScaleFactor` with `TSLRenderingAttributeTextScaleFactorUnits` of `TSLDimensionUnitsPixels` instead. If used, this attribute will force the text to be drawn with the Text Entity height attribute defining the pixel size.
- `TSLRenderingAttributeTextRotatable`: 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.

The size of the font used to render the text is calculated using the following pseudo-code:

```
if ( obsolete fixed size flag is true )
{
  sizeInPixels = Entity size
}
else
{
  switch ( textSizeFactorUnits )
  {
    case pixels :
      sizeInPixels = size factor

    case map units :
      sizeInPixels = (size factor * tmcPerMU) / tmcPerDU

    case scale factor :
      sizeInPixels = (Entity size * size factor ) / tmcPerDU
  }
}
if ( sizeInPixels < minHeight )
  sizeInPixels = minHeight ;
else if ( sizeInPixels > maxHeight )
  sizeInPixels = maxHeight ;
```

10.6.6 Symbol Rendering Attributes

These are available on Symbol Entities. They are:

- `TSLRenderingAttributeSymbolColour`: This value must be an index from the `tslcolours.dat` file, the currently loaded map palette or a 24-bit colour (see `TSLColourHelper` API Documentation). The default is –1, which inhibits display of the Entity.
- `TSLRenderingAttributeSymbolStyle`: This value must be an index from the tslsymbols.dat file. The default is –1, which inhibits display of the symbol.

Note that the icon symbols defined in the standard tslsymbols.dat file cannot currently be displayed on X11 based systems.

- `TSLRenderingAttributeSymbolSizeFactor`: 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 `TSLRenderingAttributeSymbolSizeFactorUnits`. The default is 0, which inhibits display of the Symbol.
- `TSLRenderingAttributeSymbolSizeFactorUnits`: This value is one of `TSLDimensionUnits` enum and determines how the `TSLRenderingAttributeSymbolSizeFactor` value is interpreted. Typical values allow the height of the Symbol to be defined in points, Map Units, internal TMC units or device units. An additional value for text and symbol size factors is `TSLDimensionUnitsScaleFactor`. This makes MapLink calculate the actual size of the object by multiplying the `TSLRenderingAttributeSymbolSizeFactor` by the TMC height stored on the Entity. This facility is included mainly for backwards compatibility and it is recommended that new code does not use this. However, again for backwards compatibility, the default is `TSLDimensionUnitsScaleFactor`!
- `TSLRenderingAttributeSymbolMinPixelHeight`: 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.
- `TSLRenderingAttributeSymbolMaxPixelHeight`: 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.
- `TSLRenderingAttributeSymbolRotatable`: 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.

- `TSLRenderingAttributeSymbolFixedSize`: Deprecated, use `TSLRenderingAttributeSymbolScaleFactor` with `TSLRenderingAttributeSymbolScaleFactorUnits` of `TSLDimensionUnitsPixels` instead. If used, this attribute will force the Symbol to be drawn with the Entity height attribute defining the pixel size.
- `TSLRenderingAttributeRasterSymbolScalable`: 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.
- `TSLRenderingAttributeSymbolFontCharacter`: Symbols may 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 character from the font to be displayed.

The size of the symbol used to render the text is calculated using the following pseudo-code:

```
if ( obsolete fixed size flag is true )
{
  sizeInPixels = Entity size
}
else
{
  switch ( symbolSizeFactorUnits )
  {
    case pixels :
      sizeInPixels = size factor

    case map units :
      sizeInPixels = (size factor * tmcPerMU) / tmcPerDU

    case scale factor :
      sizeInPixels = (Entity size * size factor ) / tmcPerDU
  }
}
if ( sizeInPixels < minHeight )
  sizeInPixels = minHeight ;
else if ( sizeInPixels > maxHeight )
  sizeInPixels = maxHeight ;
```

10.6.7 Raster Icon Symbols

Some of the symbols defined in the default `tslsymbols.dat` file are raster icons. These are standard windows .ico 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 `TSLRenderingAttributeRasterSymbolScalable` attribute.
- They cannot be rotated, and any rotation applied to the Symbol will be ignored.
- A .ico file may contain multiple icons. Only the first one will be used.
- The .ico file can contain icons of any size, but due to issues in the underlying Windows API, there will be a significant performance hit if either the width or height values are not multiples of 8. The transparency facility of the icon format can be used to mask out any additional pixels.
- Icon Symbols may be displayed on X11 based platforms (See the X11 Release Notes).
- The extent of the Symbol will include the full size of the icon, not just the non-masked areas.

10.6.8 Other Raster Symbols

It is possible to use other custom raster objects as symbols. 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 `TSLRenderingAttributeRasterSymbolScalable` attribute.
- Raster symbol rotation is only supported by the OpenGL 2D drawing surface. When using any other type of drawing surface rotation applied to the symbol will be ignored.
- When using raster symbols containing transparency, the TSLNTSurface and TSLMotifSurface may convert the transparency information in the image to an on/off mask.
- The extent of the Symbol will include the full size of the image, not just the non-masked areas.

10.6.9 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` and for Entities created using the Editor SDK.

Entity Based Rendering is configured using the `TSLEntity::setRendering` methods. These are a group of three overloaded methods with a single simple interface. The

methods take an enumeration defining the graphical property to set, along with the new value. There is also a parallel set of query methods.

10.6.10 Feature Based Rendering

Maps often contain lots of Entities that need to be rendered in a similar fashion. Feature Based Rendering allows the rendering styles to be defined once only for a particular map feature type and then specific Entities to be tagged with an identifier to indicate what feature type it represents. This saves memory and improves performance since the rendering styles need only be stored once and optimisations can be made to the low-level graphics calls when all features of a particular type are drawn together.

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 `TSLDrawingSurface` or on the `TSLDataLayer`. Wherever Feature Based Rendering is configured, it uses the same `setFeatureRendering` methods. These are a group of three overloaded methods with a single simple interface. The methods take the feature name, feature ID and an enumeration defining the graphical property to set, along with the new value. There is also a parallel set of query methods. Note that the feature name is optional. If NULL is passed, the feature ID is used.

10.6.11 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, 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.

10.6.12 Determining Styles and Font Indices

Many different symbols, fonts, line styles and fill styles are supplied with MapLink. The easiest way to see the available styles, and to determine their index, is to look in the MapLink Studio Feature Book.

- Start a new map project and invoke the Feature Book.
- Select the Reference Section, Layer Overview Feature.
- Each index-based property in the Feature Properties Dialog shows a sample of the current rendering, a description and the index for that style. You may need to make the Feature Properties Dialog wider to see the index – especially for some of the complex line styles which can have very long descriptions.
- Using the Feature Properties Dialog, browse the available styles to find an appropriate one and select it. Some properties display the index in the combo box used for selection. Other properties such as colour and symbol style do not display the index during selection.
- The Feature Properties Dialog will be updated to show the index of your chosen style.
- Where a particular style has multiple colours displayed, such as vector symbols or complex line styles, the configurable colour is displayed as red in the sample.

10.6.13 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`, `TSLRectangle` and `TSLBorderedPolygon`: 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.

10.6.14 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. See Section 11 for code examples of some simple rendering configurations. The Geometry Creation sample installed with MapLink gives examples of every available Rendering Attribute for each primitive type and allows you to experiment with them. Note that this sample also includes access to obsolete attributes which may clash with other newer ones!
- 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 10.6.13 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, a colour index that is not in the current palette, a line style index that does not exist in `tsllinestyles.dat`, a symbol style index that specifies an icon symbol may be illegal on X11 as are some fonts (See X11 Release Notes). Check the contents of the configuration files (see Section 12.6) or validate the styles in MapLink Studio.
- 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 `TSLRenderingAttributeVisible` and a Data Layer can be hidden using `TSLPropertyVisible`.

- 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 `TSLPropertyMinZoomDisplay` and `TSLPropertyMaxZoomDisplay` 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 `TSLPropertyMinTextHeight` and `TSLPropertyMaxTextHeight`.

11 Walkthrough 3 – Adding a Simple Vector Overlay

In this section, you will take the application that has been developed in the earlier walkthroughs and add an overlay. Within this overlay you can create simple vector objects to be displayed using Entity Based Rendering. You will also create an example of a Symbol displayed using Feature Based Rendering.

11.1 Interaction Mode Modifications

It is obvious that the existing interaction mode in the Hello Globe application is insufficient to allow complex chains of points to be built up so we will try and keep things simple. We will need to make some modifications to allow different kinds of Entities to be created without adding a lot of additional code and logic within the button handlers of the View. A further complication is that your application may be MDI and therefore have many Documents and Views open at the same time.

With these issues in mind we will make the following changes:

- Add a `TSLStandardDataLayer` to the Document.
- Add a new 'Overlays' menu with options for creating various Entities.
- Add event handlers for the menu to the Document class. The event will be sent to the currently active Document.
- Store the currently selected primitive type. This should be stored statically for MDI applications.
- In the View button handlers check for the Control key being pressed. If so, then call the associated document to create the primitive type and if successful redraw the view. Do this on release.
- In the document class, instantiate the appropriate primitive.

This is obviously a very simplified interaction and has limited encapsulation. For more complex, highly interactive primitive creation and manipulation facilities, Envitia provides the Editor SDK and the companion Spatial SDK. See later sections for details about how to integrate these into an application.

We will assume that the overlay is only available when a map is loaded.

11.2 Adding a TSLStandardDataLayer

Simple vector overlays are usually stored in a `TSLStandardDataLayer`. This will need to be added to the document class. It should be created and destroyed where appropriate and added to the `TSLDrawingSurface` when the document and view are bound together.

In the Document class definition, add a declaration of the Standard Data Layer just after the Map Data Layer:

```
TSLMapDataLayer      * m_mapDataLayer ;
TSLStandardDataLayer * m_stdDataLayer ; // This line added
```

The new class variable should be initialised to `NULL` in the Document constructor.

```
CHelloGlobeDoc::CHelloGlobeDoc()
  : m_mapDataLayer( NULL ), m_stdDataLayer( NULL )
{
}
```

The overlay layer should only be created after a map has been successfully loaded and should be destroyed when the map layer is destroyed.

In the Document `OnOpenDocument` method, instantiate a `TSLStandardDataLayer` if the map is successful:

```
if ( !m_mapDataLayer->loadData( lpszPathName ) )
{
  // Error handling as before
  return FALSE ;
}
m_stdDataLayer = new TSLStandardDataLayer() ; // This line added
```

In the Document `DeleteContents` method, add the following code to delete the overlay layer:

```
if ( m_stdDataLayer )
{
  m_stdDataLayer->destroy() ;
  m_stdDataLayer = NULL ;
}
```

Modify the Document `addToSurface` method to add the extra layer:

```
if ( !m_mapDataLayer || !m_stdDataLayer || !drawingSurface )
  return false ;
bool sts = drawingSurface->addDataLayer( m_mapDataLayer, "map" ) ;
if ( sts )
  sts = drawingSurface->addDataLayer( m_stdDataLayer, "overlay" ) ;
return sts ;
```

11.3 Adding the Overlay Menu and Handlers

This menu will allow the user to select which type of Entity will be created on a button press.

Use the Dev Studio Resource editor to create a menu called Overlays, with options for Text, Symbol, Polygon, Line and Feature.

You may also wish to add toolbar icons to invoke the menu items.

We need somewhere to store the current overlay type selection. For MDI applications, we will store this globally to avoid confusing the user when swapping between currently open documents. Initialise it to the ID of one of the Overlay menu items.

In the Document class definition, add a declaration for a static integer to hold the currently selected primitive and initialise it appropriately in the Document .cpp file

```
// This line added in the Document class header, private section
static int m_overlayType ; // This line added in class header

// This line added in the Document .cpp file
int CHelloGlobeDoc::m_overlayType = ID_OVERLAYS_TEXT ;
```

Now we need to add `COMMAND` handlers to update the chosen overlay type on a user selection.

Use Class Wizard to add `COMMAND` handlers for the overlay menu items to the Document and in each handler set the `m_overlayType` variable. You could use a range command handler and only have one method, but for simplicity we have added one per menu item. You should also add `UPDATE_COMMAND_UI` handlers to provide some feedback to the user about which overlay type is selected. The handlers for Polygons are shown below. Add them for each of the menu entries

```
void CHelloGlobeDoc::OnOverlaysPolygon()
{
   m_overlayType = ID_OVERLAYS_POLYGON ;
}

void CHelloGlobeDoc::OnUpdateOverlaysPolygon(CCmdUI *pCmdUI)
{
  pCmdUI->SetCheck( pCmdUI->m_nID == m_overlayType ) ;
}
```

Try building your application. At this point you should have the user interface working, but no primitive creation happening. Check that the `m_overlayType` variable is set correctly when you select each menu item or toolbar button and that the menu items are ticked correctly.

11.4 Adding the Overlay Creation Interface

With the user interface working, we now need to add the back-end methods which create the Entities. These will be triggered from the View, but to maintain encapsulation should actually be in the Document.

Add a public method to the Document, called `createOverlay`. This should return a boolean value which indicates whether the creation was successful. As a parameter it will take the Drawing Surface and the position at which the overlay should be created.

Create other, private methods, which provide the implementation for creating each overlay type. These will have the same signature as `createOverlay` and should be called from it according to the current value of `m_overlayType`. The code fragment below shows the implementation of `createOverlay` and a dummy implementation of the text method.

```
bool CHelloGlobeDoc::createOverlay(long x,long y,TSLDrawingSurface *ds)
{
  switch ( m_overlayType )
  {
    case ID_OVERLAYS_LINE :    return createPolyline(x, y, ds) ;
    case ID_OVERLAYS_POLYGON : return createPolygon(x, y, ds) ;
    case ID_OVERLAYS_TEXT :    return createText(x, y, ds) ;
    case ID_OVERLAYS_SYMBOL :  return createSymbol(x, y, ds) ;
    case ID_OVERLAYS_FEATURE : return createFeature(x, y, ds) ;
  }
  return false ;
}
bool CHelloGlobeDoc::createText(long x,long y,TSLDrawingSurface*ds)
{
```

11.5 Triggering the Overlay Creation

In the View class `LButtonUp` handler, we should call the `createOverlay` method if the Control button is pressed.

```
void CHelloGlobeView::OnLButtonUp(UINT nFlags, CPoint point)
{
  if ( m_drawingSurface )
  {
    if ( nFlags & MK_CONTROL )
    {
      TSLTMC x, y ;
      m_drawingSurface->DUToTMC( point.x, point.y, &x, &y ) ;
      if ( GetDocument()->createOverlay( x, y, m_drawingSurface ) )
          InvalidateRect( 0, FALSE ) ;
    }
    else if ( abs( . . . . /* Rest of method the same */ ) )
```

11.6 Creating the Text Overlay

Build the application and ensure that the create methods are being triggered when the Control – Left Mouse Button combination is used.

Now we will create some text at the button click location and if successful return true. The success return status will force the View to redraw itself.

Replace the dummy implementation of `createText` with the following:

```
TSLEntitySet * es  = m_stdDataLayer->EntitySet() ;
TSLText      * txt = es->createText( 0, x, y, "Hello World" ) ;
if ( !txt )
  return false ; // Return failure if text could not be created

TSLStyleID black = TSLDrawingSurface::getIDOfNearestColour( 0, 0, 0 ) ;

// Set the rendering of the text to be black, Arial, 25 pixels high
txt->setRendering( TSLRenderingAttributeTextFont, 1 ) ;
txt->setRendering( TSLRenderingAttributeTextColour, black ) ;
txt->setRendering( TSLRenderingAttributeTextSizeFactor, 25.0 ) ;
txt->setRendering( TSLRenderingAttributeTextSizeFactorUnits,
                   TSLDimensionUnitsPixels ) ;

// Tell the layer that its contents have changed
m_stdDataLayer->notifyChanged( true ) ;

return true ;
```

The key points of this code are

- The Text Entity is created in the Entity Set of the overlay Standard Data Layer. If the static `TSLText::create` method was used, then the Text Entity would not be attached to any Drawing Surface and hence would never be displayed.

- The `createText` method of the Entity Set returns false if the creation failed. If this happens, examine the contents of the error stack to determine why. With Text, it is usually because the string is empty.
- You can lookup colour indices in the currently loaded palette using the static `TSLDrawingSurface::getIDOfNearestColour` method. Note: You may wish to use 24bit RGB colour (see `TSLColourHelper` API Documentation).
- Ensure that you set the minimum attributes required, as specified in section 10.6.13. Other attributes are optional.
- Always call `notifyChanged` after completing a sequence of modifications to the layer. If this is not called, then MapLink will assume that any buffer associated with the Drawing Surface or Data Layer are up to date until the next change of viewing area.

11.7 Creating the Symbol Overlay

The example Symbol that we shall create will be sized in Map Units. This means that it will scale according to the current zoom factor. It is very dependent upon the current map loaded. We are assuming that the sample Dorset map is loaded. This may be found in:

`<MAPLINK_HOME>\Samples\MapForSamples\Dorset\Dorset.map`

Replace the dummy implementation of `createSymbol` with the following:

```
TSLEntitySet * es = m_stdDataLayer->EntitySet() ;

TSLSymbol * symbol = es->createSymbol( 0, x, y ) ;
if ( !symbol )
  return false ;

// Create a green star, 1000m high.
// This looks sensible on the Dorset map!
TSLStyleID green= TSLDrawingSurface::getIDOfNearestColour( 0, 255, 0 ) ;
symbol->setRendering( TSLRenderingAttributeSymbolStyle, 14 ) ;
symbol->setRendering( TSLRenderingAttributeSymbolColour, red ) ;
symbol->setRendering( TSLRenderingAttributeSymbolSizeFactor,1000.0);
symbol->setRendering( TSLRenderingAttributeSymbolSizeFactorUnits,
                      TSLDimensionUnitsMapUnits ) ;

// Tell the layer that its contents have changed
m_stdDataLayer->notifyChanged( true ) ;

return true ;
```

Many different symbols are supplied with MapLink. Use the method described in section 10.6.12 to determine an appropriate index.

11.8 Creating the Polygon Overlay

Unlike Text and Symbol Entities, Polygons are defined with many coordinates defining the boundary of the polygon. This list of coordinates must be created via the `TSLCoordSet` class and passed to the Polygon create method.

The example below uses the Drawing Surface conversion routines to determine the current zoom factor and then creates a polygon at a specific screen size for the current zoom factor. Zooming in or out will make the Polygon appear larger or smaller.

Replace the dummy implementation of `createPolygon` with the following:

```
    TSLEntitySet * es = m_stdDataLayer->EntitySet() ;

    // Create a coordinate list forming a triangle around the position
    // Use the Drawing Surface to calculate the coordinates
    // We will make our triangle 25 pixels either side of the position
    // Note that the pixels are at the current zoom factor - the polygon
    // is always completely scalable
    TSLCoordSet * coords = new TSLCoordSet() ;
    if ( !coords )
      return false ;

    double tmcPerDU = ds->TMCperDU() ;
    coords->add( x - 25 * tmcPerDU, y - 25 * tmcPerDU ) ;
    coords->add( x + 25 * tmcPerDU, y - 25 * tmcPerDU ) ;
    coords->add( x,                 y + 25 * tmcPerDU ) ;

    // Hand ownership of the coordset to the polygon
    TSLPolygon * poly = es->createPolygon( 0, coords, true ) ;
    if ( !poly )
      return false ;

    TSLStyleID yellow = TSLDrawingSurface::getIDOfNearestColour( 255, 255, 0
);
    TSLStyleID black  = TSLDrawingSurface::getIDOfNearestColour( 0, 0, 0 ) ;
    poly->setRendering( TSLRenderingAttributeFillStyle, 1 ) ;
    poly->setRendering( TSLRenderingAttributeFillColour, yellow ) ;
    poly->setRendering( TSLRenderingAttributeEdgeStyle, 1 ) ;
    poly->setRendering( TSLRenderingAttributeEdgeColour, black ) ;
    poly->setRendering( TSLRenderingAttributeEdgeThickness, 1.0 ) ;

    // Tell the layer that its contents have changed
    m_stdDataLayer->notifyChanged( true ) ;

    return true ;
```

11.9 Creating the Polyline Overlay

Polylines use the same mechanism as Polygons to specify the list of coordinates that should be used.

The example below uses the Drawing Surface conversion routines to determine the current zoom factor and then creates a Polyline at a specific size for the current map – i.e.

the Polyline is drawn in real-world units. It also uses the Thickness Units Rendering Attribute to specify the thickness of the line in real-world units. Zooming in or out will make the Polygon appear larger or smaller and change the displayed thickness of the line. The values chosen are appropriate for the sample Dorset map.

Replace the dummy implementation of `createPolygon` with the following:

```
TSLEntitySet * es = m_stdDataLayer->EntitySet() ;

// Create a coordinate list forming a triangle around the position
// Use the Drawing Surface to calculate the coordinates
TSLCoordSet * coords = new TSLCoordSet() ;
if ( !coords )
  return false ;

// Make a triangle, 1km either side of the specified position
double tmcPerMU = ds->TMCperMU() ;
coords->add( x - 1000 * tmcPerMU, y + 1000 * tmcPerMU ) ;
coords->add( x,                   y - 1000 * tmcPerMU ) ;
coords->add( x + 1000 * tmcPerMU, y + 1000 * tmcPerMU ) ;

// Hand ownership of the coordset to the polygon
TSLPolyline * poly = es->createPolyline( 0, coords, true ) ;
if ( !poly )
  return false ;

// Use MapUnit thickness so the line thickness scales as we zoom
// in/out
// Set it to 20m. Complex line styles - like style 48 have thickness
// clamping applied automatically to avoid performance
// or aesthetic problems
TSLStyleID yellow = TSLDrawingSurface::getIDOfNearestColour(255, 255, 0) ;
poly->setRendering( TSLRenderingAttributeEdgeStyle, 48 ) ;
poly->setRendering( TSLRenderingAttributeEdgeColour, yellow ) ;
poly->setRendering( TSLRenderingAttributeEdgeThickness, 20.0 ) ;
poly->setRendering( TSLRenderingAttributeEdgeThicknessUnits,
                    TSLDimensionUnitsMapUnits) ;

// Tell the layer that its contents have changed
m_stdDataLayer->notifyChanged( true ) ;

return true ;
```

11.10 Creating the Feature Based Symbol Overlay

The previous examples have used Entity Based Rendering, in which every Entity has its own definition of Rendering Attributes. As discussed in Section 10.6.10, MapLink also has the capability to specify the Rendering Attributes of a Feature type on the Data Layer or Drawing Surface and on the individual Entity specify which Feature the Entity represents.

To do this, we must first configure the Data Layer and specify the rendering.

After the Standard Data Layer has been constructed, define some Feature Rendering for use during the Feature creation and rendering.

In the Document `OnOpenDocument` method, add the following code:

```
TSLStyleID black = TSLDrawingSurface::getIDOfNearestColour( 0, 0, 0 ) ;

// Make up a feature name and numeric ID
m_stdDataLayer->addFeatureRendering( "Airport", 123 ) ;

// Associate some rendering with the new feature, use ID for
// efficiency
m_stdDataLayer->setFeatureRendering( 0, 123,
                TSLRenderingAttributeSymbolStyle, 6003 ) ;
m_stdDataLayer->setFeatureRendering( 0, 123,
                TSLRenderingAttributeSymbolColour, black ) ;
m_stdDataLayer->setFeatureRendering( 0, 123,
                TSLRenderingAttributeSymbolSizeFactor, 40.0 ) ;
m_stdDataLayer->setFeatureRendering( 0, 123,
                TSLRenderingAttributeSymbolSizeFactorUnits,
                TSLDimensionUnitsPixels);
```

Next, we implement the `createFeature` method to create the symbol referencing the Feature ID that we have just created.

Replace the dummy implementation of `createFeature` with the following:

```
TSLEntitySet * es = m_stdDataLayer->EntitySet() ;

// 123 is the numeric feature code we assigned on the Data Layer
TSLSymbol * symbol = es->createSymbol( 123, x, y ) ;

if ( !symbol )
  return false ;

// No need to configure any rendering, MapLink will look it up
// from the Data Layer at display time.

// Tell the layer that its contents have changed
m_stdDataLayer->notifyChanged( true ) ;

return true ;
```

Congratulations! You can now create vector overlays!

12 More Features of the Core SDK

This section discusses features of the Core SDK above and beyond simple display and creation of maps.

12.1 Coordinate Systems

The class `TSLCoordinateSystem` encapsulates the transforms used to create a map. A coordinate providing layer provides a `TSLCoordinateSystem` to allow the user to convert between latitude/longitude, Map Units (MU) and TMCs. A `TSLCoordinateSystem` can be created by the developer and used to convert between different Map Projections.

More information on Coordinate Systems and Map Projections can be found in the "MapLink Studio Users Guide" and Help.

12.1.1 Transverse Mercator

EPSG have changed the formula used for Transverse Mercator while retaining the EPSG IDs for the affected Coordinate Systems that use Transverse Mercator.

The original formula "USGS Snyder" and "JHS" formulas produce similar results in a +-4 degree band around the central longitude. Outside this band the results diverge. The JHS algorithm is more accurate out to +-40 degrees.

EPSG recommend that the two formulas are not mixed.

EPSG recommend the use of the JHS formula.

The Snyder formula was used in MapLink 7.5 and older versions. Both formulas are available in MapLink 8.0 and newer.

To address the EPSG change to the Transverse Mercator several changes have been made to `tsltransforms.dat` that may affect an application. The changes are outlined below:

- MapLink coordinate system IDs in the range [-5000..-9000] use the Transverse Mercator JHS projection algorithm.
- IDs in the range [-1..-4900] use the original USGS Snyder Transverse Mercator projection algorithm.
- The EPSG ID in the case of both Coordinate Systems are the same.
- The NAME has been updated to contain '`(Snyder)`' or '`(JHS)`' to distinguish the algorithm used.

Where:

ID is the value used in `TSLCoordinateSystem::findByID()` and returned by `id()`.

NAME is the value used by `TSLCoordinateSystem::findByName()` and returned by `name()`.

The method `TSLCoordinateSystem::findByName()` functionality has changed slightly.

When looking up a Coordinate System that uses Transverse Mercator projection the method expects one of two forms to be used, for example:

- `"UTM (WGS84) Zone 1 North (Snyder)"`
- `"UTM (WGS84) Zone 1 North (JHS)"`

For backwards compatibility the `findByName()` method will return the original Snyder Coordinate System if `"(Snyder)"` or `"(JHS)"` is missing from the name being searched for. In this case a warning will be placed on the `TSLThreadedErrorStack`.

The method `TSLCoordianteSystem::findByEPSG()` may not work as expected, for example;

```
const TSLCoordinateSystem *coordSystem =
TSLCoordianteSystem::findByEPSG(27700);
```

Returns the OSGB coordinate system using the Snyder Transverse Mercator formula. For the new formula you need to do the following:

```
const TSLCoordinateSystem *coordSystems[2];
int numberFound = TSLCoordianteSystem::findByEPSG(27700, coordSystems, 2);
```

You would need to check the `numberFound` variable and then validate the name of each returned `TSLCoordinateSystem` to see if it was the Snyder or JHS version. You could use the MapLink IDs as these are unique.

12.1.2 TSLCoordinateConvertor

The class `TSLCoordinateConvertor` can be used to convert between latitude/longitude, GARS, MGRS, UTM and UPS.

Additionally this class contains methods that do the following conversions:

- Great Circle
- Vincenty
- Rhumbline
- Geocentric
- Geodetic
- Topocentric

12.2 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`, thus allowing the same Data Layer to be displayed differently on two different surfaces – e.g. in an overview pane and the main window.

A Drawing Surface contains a list of decluttering settings for each Data Layer in the surface, and an additional decluttering list that applies to all Data Layers (sometimes referred to as the global decluttering list). The decluttering list for each Data Layer inherits

the contents of the global decluttering list, thus allowing decluttering to be easily set up for several Data Layers in one method call, while still allowing decluttering settings to be overridden on a per-Data Layer basis.

12.2.1 Declutter Feature Name and ID

The declutter status is configured on a per-feature basis using the Feature 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 period 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 ID's. 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.

12.2.2 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`, `TSLDeclutterStatusOff` or `TSLDeclutterStatusAuto`. To set the status use:

```
m_drawingSurface->setDeclutterStatus( "feature", status, layer )
```

The layer argument is optional. If specified, it targets the decluttering at a specified Data Layer. Otherwise, the feature will be decluttered on all data layers in the drawing surface through the Drawing Surface's global decluttering list. Decluttering a specific data layer is sometimes desirable since a Drawing Surface may contain multiple data layers containing the same features, and the application may wish to only declutter the feature from a single data layer.

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 `TSLDeclutterStatusResultPartial`, 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.

12.2.3 Automatic Decluttering on Zoom

In addition to the standard On and Off declutter status values, it is possible to set the status to be `TSLDeclutterStatusAuto`. 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.

12.2.4 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 all data layers in a drawing surface with the default Feature Name, use the following call:

```
ds->setDeclutterStatus("Rasters",TSLDeclutterStatusOff);
```

12.3 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.

12.3.1 Finding the Entity under the Cursor

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.

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 in an editing environment.
- Any Data Layers with the `TSLPropertyDetect` property set to false and any Entity with the `TSLRenderingAttributeSelectable` attribute set to false will be ignored in the search. Note that the default value for `TSLPropertyDetect` 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.

12.3.2 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 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, 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 `TSLRenderingAttributeSelectable` 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 `TSLSelectorActionSelectNext` 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. They are references to the real Entities and so should not be destroyed. The application can iterate through the references in the `TSLMapQuery` object to further act upon the Entities.

12.3.3 Picking

'Picking' may be performed via the 'pick' method. This allows selection of all types of object, including 2D Entities, 3D Entities, Display Objects, Satellites and even custom objects.

The `TSLDrawingSurfaceBase` provides 2 pick methods; both take a pixel location, aperture and an optional `TSLPickSelector` parameters, while one also takes a data layer name parameter. The `TSLPickSelector` is an abstract class that allows users to filter the results based upon their own criteria. The data layer name parameter is used to first filter the results to only include those from a desired data layer.

The return value of pick operations is an instance of the `TSLPickResultSet` class which controls a set of `TSLPickResult` objects. This `TSLPickResult` class is an abstract wrapper around the actual object found in the query location. The Core SDK provides derivatives such as `TSLPickResultEntity` and `TSLPickResultCustom`, while other SDKs provide additional derivatives. A user can determine the correct cast required by calling the `queryType` operation on the object or the `isType` static operations on the derivative types.

If a user wishes to provide their own custom pick result type from their `TSLClientCustomDataLayer`, then they should derive a class from the abstract `TSLClientCustomPickResult` type. In the pick method of their `TSLClientCustomDataLayer` derivative class, they should add instances of this new class to the passed `TSLPickResultSet` object.

12.3.4 Other Searching Facilities

There are several older searching methods (`findEntityXXX`) available on the `TSLDrawingSurface` and `TSLDataLayer`, but these have very specific rules which are detailed in the method descriptions. See the API Class Documentation for further details.

12.4 Dynamic Rendering

The core MapLink SDK provides a mechanism by which users can dynamically alter how data is rendered without modifying the data. This can be performed globally on all data in a drawing surface or locally for selected data layers.

In order to take advantage of this technique, a class derived from `TSLClientCustomDynamicRenderer` should be created. See the Dynamic Rendering sample for example code. MapLink includes a premade dynamic renderer for S52 rendering. Future MapLink releases may provide standard dynamic renderers for such actions as Thematic mapping.

When a data layer is drawn with an active dynamic renderer, the `render` method of the `TSLClientCustomDynamicRenderer` will be called for each entity that is being drawn. The dynamic renderer determines how the entity will be drawn - it can ask MapLink to draw the entity as normal, draw the entity with different rendering attributes or perform its own custom drawing instead.

When using a dynamic renderer to draw an entity using different rendering attributes, the dynamic renderer can either call the relevant `setupAttributes` methods and then call the appropriate draw method (e.g. `drawPolygon`) on the `TSLRenderingInterface`, or call the `setupAttributes` methods and return `TSLDynamicRendererActionUseCurrentRendering` from the dynamic renderer's `render` method. Both approaches will result in the same output, however returning

`TSLDynamicRendererActionUseCurrentRendering` from the dynamic renderer is more efficient in some cases and so is the preferred method in this situation.

12.5 Optimisation Techniques

MapLink is targeted to high-performance applications. This section describes a few techniques available to increase the performance and responsiveness of an application.

12.5.1 Buffering

When an application draws directly to the screen, two things become obvious:

- Flicker makes it apparent that the drawing is taking place, since the screen blanks and then becomes populated with the picture. Even with a high-performance graphics card, it does not require much data to be displayed before this begins to detract from the aesthetics of an application.
- The second thing that is apparent is that every layer is drawn, even if it hasn't changed. This is especially obvious in systems with moving objects over a static map.

MapLink has several features to address these issues, through the use of buffering at different levels. The problem of flicker may be reduced using Drawing Surface buffering (back buffering). Once this has been configured, MapLink draws primitives into an off-screen buffer and once complete, copies the off-screen buffer to the screen. If nothing has changed since the last draw, then the existing buffer is copied. MapLink will automatically flag the back buffer as invalid when the visible map area has changed, or the Drawing Surface has been resized.

To configure a Drawing Surface as buffered add the following code to your application, usually just after the Drawing Surface has been created.

```
m_drawingSurface->setOption( TSLOptionDoubleBuffered, true ) ;
```

Back buffering is sufficient where only a single Data Layer is being displayed or in situations where the contents of all attached Data Layers change at the same time. In many applications, there are multiple Data Layers with a set of fairly static underlays and at least one dynamic or editable overlay. For these applications, MapLink has secondary buffering. This is associated with a group of Data Layers that are attached to the same Drawing Surface. In this type of buffering, all Data Layers in the group are drawn into a separate off-screen buffer before being copied to the back buffer or screen. Any non-buffered Data Layers are then drawn. If no buffered Data Layers have changed, then the existing secondary buffer is copied without being redrawn.

To configure a Data Layer to be part of the buffered group for a particular Drawing Surface, add the following code to your application, usually just after the Data Layer has been added to the Drawing Surface.

```
m_drawingSurface->setDataLayerProps( "layername",
                                     TSLPropertyBuffered, 1 ) ;
```

In MFC based applications that use buffering, you should handle the `WM_ERASEBGRD` event in order to stop Windows clearing the window itself and thus introducing flicker. Typically,

just override the `OnEraseBackground` method in the applications View object and return True. This is appropriate when MapLink is rendering to the entire window. In situations where MapLink is only drawing to part of the window then the application should clear the parts of the window that MapLink does not render.

12.5.2 Tiled Buffering

Some drawing surfaces offer a more advanced version of data layer buffering called Tiled Layer Buffering. This extends the buffering concept to split the total extent of all layers in the drawing surface into a grid of tiles, which are rendered asynchronously as needed. This means that in contrast to regular buffered layers, tiled buffering means that the buffered layers do not need to be redrawn when panning the view.

Tiled buffering can be enabled by adding the following code to your application, usually just after the Drawing Surface has been created:

```
m_drawingSurface->setOption( TSLOptionTileBufferedLayers, true ) ;
```

As the buffered tiles are generated asynchronously, the application will be notified when new tiles are available through the `TSLDrawingSurfaceDrawCallback`.

When zooming using tiled buffering, MapLink will not block the application waiting for the buffered tiles to be redrawn. This means that buffered layers will not be drawn until the tiles for the new viewing resolution are ready. Since this is often undesirable, Progressive Zooming can optionally be enabled using the following code:

```
m_drawingSurface->setOption( TSLOptionProgressiveTileZoom, true ) ;
```

This option instructs the drawing surface to use previously created tiles to fill in for tiles at the new zoom scale that are not currently ready for drawing, meaning that buffered layers will still be visible when zooming.

More information on tiled buffering can be found in the API documentation for the `TSLDrawingSurfaceTiledBufferControl` class.
Tiled buffering is currently supported by the OpenGL drawing surface.

12.5.3 Caching

Within MapLink Studio, there is much emphasis on efficient tiling and layering of a map to ensure that the run-time application can optimise the performance. This means that in a system with a moving map, or where the user is zooming and panning, that map tiles are being loaded and destroyed. This is itself an obvious performance hit.

12.5.4 Memory Cache

To help mitigate the performance hit of tile swapping, the MapLink Core SDK has facilities to configure an in-memory cache of recently used tiles. To avoid swamping low-spec machines, the default cache configuration is fairly low at 32Mb. This can be set on each Map Data Layer or Raster Data Layer using the following method:

```
m_mapDataLayer->cacheSize( sizeInKiloBytes ) ;
```

The cache may be further configured using the `cacheFlushLimit` which is the number of tiles that the Data Layer attempts to keep in memory when it overflows. Note that when a

tile is added to the cache, its size is assumed to be the same as the disk size – except for compressed raster images which are expanded on loading. If a tile has been saved from MapLink Studio using the Enterprise Compression or Optimised for Size options, then there will be some level of expansion in memory and you should account for this when setting your cache size.

If your map appears fast in the MapLink Viewer, but slow in your application then your memory cache size may be the problem.

12.5.5 Persistent Cache

The Map Data Layer also has the facility to store tiles in a secondary, disk-based persistent cache. This is typically used when employing the Remote File Loader so this topic is covered in that section. Basic information may be found in the online help for the `TSLPersistentCacheData` object.

12.6 Rendering Configuration Files

Many Rendering Attributes define an integer index into configuration files. These configuration files must be read at the start of a MapLink application as described in Section 8.3. There are 5 basic files. This section describes the contents and format of the current versions. When loaded, these files are used across the MapLink application – only one version of each configuration file may be loaded at any one time. You should note that MapLink is consistently in development so if you choose to modify the standard files then you may not be able to take advantage of any future enhancements.

12.6.1 Colours

The colours file, usually called tslcolours.dat, holds the definition of the colour palette and associated RGB values. The format is quite simple and is identical to the palette file written out alongside a map by MapLink Studio. The first few lines of a colours file are shown below. Each line is commented in red – although the comments should not appear in the actual file.

```
TSLCL 105                    // File ident and format version number
ColourCubePalette            // Name of palette, for MapLink Studio
;                            // Field separator on subsequent lines
1;184;134;11;Dark Goldenrod // Index;Red;Green;Blue;Name in FeatureBook
2;189;183;107;Dark Khaki     // Next colour ... and so on 203 times
```

By default, maps generated from MapLink Studio have an associated palette file. This palette file will be loaded by the MapLink SDK when the map is loaded. This means that the global palette may change. For this reason, it is recommended that all maps to be loaded into an application are constructed using the same palette.

A number of colour index values are reserved as follows:

- MapLink Studio User defined colour base: 500
- Symbols colour range: 100000..100023
- S-52 colour range: 110000..110200
- Dynamic colour range: 120000..130000

12.6.2 Line Styles

The line styles file, usually called tsllinestyles.dat, holds the definition of the edge styles used for polylines, arcs and the boundaries of surfaces such as polygons. The format is rather more complex than the colours file.

There are several different types of line style, some more complex than others. The example below contains one of each currently supported type; these are explained later.

```
TSLES 108                    // File ident and format version number
;                            // Field separator on subsequent lines
#This is a comment
I;10;99;linestyles/anotherlinestylesfile.dat
#Above is an include declaration to another file.
S;This is a section heading   // Section name for subsequent styles
1;standard;Solid;0;3;1;0;0;0
2;standard;Dash (cosmetic pen);1;3;1;0;0;0
3;standard;Dot (cosmetic pen);2;3;1;0;0;0
4;standard;Dash-dot (cosmetic pen);3;3;1;0;0;0
5;standard;Dash-dot-dot (cosmetic pen);4;3;1;0;0;0
6;standard;Dash 8;6;3;1;0;1;4 8 8 8 8
9;multi;Narrow road, light casing;2;[1,(153,153,153),3];[1,(-1,-1,-1),1]
10615;ttlclsstrk;Waterfall;GeoSym0615;C[(100,100,200),1]D[0,0]D[10,0]
```

- Versions 107 and newer versions of the file must be saved in the UTF-8 code page without a BOM (Byte Order Mark).
- Some index values are reserved. Please refer to the file for additional information.
- Comments appear prefixed by the `#` character and are ignored by MapLink.

Include declarations allow for a more structured layout of the line style file by segregating different categories of style into their own file.
The declaration is prefixed by the '`I`' character followed by the lower and upper ranges of the styles they contain.
The final semi-colon delimited value is filename relative to the current file. The format of this sub-file is the same as the root file and can in turn include other files. Should the file not be found at the location indicated, any associated `TSLPathList` will be checked instead.
One of the major benefits, also introduced, of using included sub-files is that these files can be delay-loaded or in other words only loaded when they are required.

- Section headings are a concept borrowed from the symbol lists and allows the categorisation of styles from the point of view of the run-time. They appear in the file prefixed with '`S`' followed by the section name, such as 'APP6A'.

Every style that appears after a section heading declaration will be associated with that section, although this does not include styles that appear in sub-files included within the section. Sub-files require their own section heading. Section headings may appear more than once within the tree of files with all styles that appear

under each of these section heading declarations being associated with the same section.
The SDKs now allow the querying of a style's associated section using the `getXxxStyleValue` methods.

- In order to make it easier for custom styles to be managed, a specific 'user' sub-file with a specific range of index values has been defined and appended to the end of the line styles file. This is called `'linestyles\tsllinestylesuser.dat'`.

This file is not shipped by default with MapLink, but it's non-existence will not generate an error.
Custom lines styles should be added to this file, thus making it easier to manage user-defined styles across MapLink upgrades.

The remaining entries in the file begin with the line style index (used with `TSLRenderingAttributeEdgeStyle`), the type (`standard/multi/dllname`) and a textual description that is displayed in MapLink Studio Feature Book. The rest of the fields for each entry are type dependant.

12.6.3 Standard Linestyles

Standard Operating System: These are of various types indicated by the first type specific field.

- Type 0: Solid.
- Type 1: Dashed.
- Type 2: Dotted.
- Type 3: Dash-Dot pattern.
- Type 4: Dash-Dot-Dot pattern.
- Type 5: NULL (invisible) pen
- Type 6: User defined pattern
- Type 7: Alternating on-off pixels (Win 2000 and XP only)

The subsequent fields are defined below:

- Obsolete: Ignored from MapLink 4.5 onwards
- Join style: 0=Bevel, 1=Mitre, 2=Round, geometric only.
- Obsolete: Ignored from MapLink 4.5 onwards
- Geometric: To indicate a cosmetic (0) or geometric (1) pen
- Pattern size: For user-defined patterns, geometric only

User defined patterns may only be applied to geometric pens and are a sequence of on-off pairs of device unit values, each value separated by spaces. The first value in the sequence is a count of such values. On Windows platforms, cosmetic pens use efficient operating system specific methods of rendering, but are limited to single-pixel wide

patterned lines. Attempting to draw a wide cosmetic line will force the style to solid. This is an operating system function, e.g.

```
27;standard;Dash 4, very wide spacing;6;3;2;0;1;4 4 16 4 16
```

12.6.4 Multi-pass Linestyles

Basically, multi-pass line styles are created by using a combination of any other line style in the tslsymbols.dat file to build up a more complex style. This allows you to specify two simple styles and then use the 'multi-pass' functionality to add them together. Each pass draws a different line style on top of the previous one which can be used to build up the desired effect.

Take the following example:

> Assume we want to create a line consisting of a thin red line in the centre of a thick black line. Essentially, we want to draw a 3-pixel high black line first and then a 1 pixel high red line over the top. (Note that this style will always be no less than 3 pixels high, even if the user sets it to 1 pixel).

To create this style:

1) Open the tsllinestyles.dat (This can be found in your MapLink/config directory. It is probably better if you back it up before you edit it).

2) Scroll to the bottom of the file and enter:

   ```
   X;multi;Line Style Description;2;[1,(0,0,0),3];[1,(255,0,0),1]
   ```

   The '2' shows that the line has two token values.
   The token '`[1,(0,0,0),3]`' says "use the solid line style (1), draw in black (0,0,0) and three units wide (3)".
   The token '`[1,(255,0,0),1]`' says "use the solid line style (1), draw in red (255,0,0) and 1 units wide (1)".

If you use (-1,-1,-1) in the token for the colour value, the user will be able to override the colour at runtime (i.e. it can be any colour) using the `TSLRenderingAttributeEdgeColour` or `TSLRenderingAttributeExteriorEdgeColour`.

The 'X' should be replaced by a unique number, but you must enter 'multi' as the second value

12.6.5 Stroked Linestyles

Stroked linestyles are implemented by an extension shared library (`ttlclsstrk`). The shared library is written specifically for the target platform.

The file tsllinestyle.dat contains many stroked linestyle definitions. An example of a stroked linestyle is shown below (note that this should all be on a single line, but is split in this document for clarity):

```
1000;ttlclsstrk;My Custom Line;MyCustomLineStyleTag;
C[(-1,-1,-1),4]U[0,-2]D[5,0]D[5,0]B
C[(-1,-1,-1),2]U[0,1]D[5,0]D[5,0]D[4,0]
```

The above line is broken down as follows:

```
StyleID;typeOrCustomDLLName;Textual comment displayed in
feature book, no semi-colons allowed;DLL specific information
```

For the `ttlclsstrk.so/sl/dll` (DLL) which implements this type of line, the DLL specific information is:

```
UniqueTag;CommandString
```

The CommandString is a chain of

`C[(R,G,B),W]`	Colour and width, RGB (red, green blue), W width. If R, G and B are all -1, then colour defined by `TSLRenderingAttributeEdgeColour` or Feature Book is used
`D[x,y]`	Pen down, line to (currentPositionX + x, currrentPositionY + y)
`U[x,y]`	Pen up, move to (currentPositionX + x, currrentPositionY + y)
`B`	Bend Point. This is where we can effectively start a new line segment.

Where:

Pen Down means place the drawing point on the paper and draw to the specified position from the current position.

Pen Up means raise the drawing point off the paper and move to the specified position from the current position.

All moves are relative to the current position.

The easiest approach when creating or modifying a Stroked Linestyle is to use a pen and a piece of graph paper, recording exactly how you draw the line (pen up, pen down, colour, amount moved).

So for '`D[5,0]U[5,0]D[5,0]`', you get the following simple line:

```
'-----  -----'
```

Where:

-	represents pen colour being drawn (Pen Down).
space	represents pen colour not being drawn (Pen Up)

The start point of your line is always at position [0, 0].

In the above simple line at the end of the sequence the current drawing position is [0, 15].

So if you wanted to return to [0, 0], you would append 'U[0,-15]' to the line definition.

Please note the following:

- When drawing a stroked linestyle MapLink uses the horizontal axis where y=0, as the middle of the line.
- A style must progress along the x-axis.
- The line thickness is specified in pixels. So a line thickness of three will be drawn in a similar way that Windows/X11 will draw a solid line of thickness 3.

- Increasing the thickness of the line via Rendering Attributes or Feature Book settings will extend the vertical axis of the stroke definition but will not extend the horizontal axis.
- Line segments are drawn with a round end cap on windows.
- You can also increase the number of 'B's to improve the ability of the custom line style to follow the draw points.
  In general 'B' points must occur when the y-axis is at 0. If you make changes to a linestyle check the changes using a relatively complex map or drawing.
- Stroked linestyles will have an impact on drawing performance. The more complex a linestyle the larger the impact on performance.
- The stroked linestyle needs to be put in the `tsllinestylesuser.dat` file.
- If you add any linestyles use the style id's `50000-59999`.

12.6.6 Fill Styles

The fill styles file, usually called `tslfillstyles.dat`, holds the definition of the fill styles used for surfaces such as polygons, rectangles and ellipses. The format is similar to the line styles file.

There are several different types of fill style, some more complex than others. The example below contains one of each currently supported type; these are explained later. Each line is commented in red – although the comments should not appear in the actual file.

```
TSLFS 106                    // File ident and format version number
;                            // Field separator on subsequent lines
#This is a comment
I;10;499;fillstyles/anotherfillstylesfile.dat
#Above is an include declaration to another file.
S;This is a section heading   // Section name for subsequent styles
1;standard;Solid;4;0;0
2;standard;Wide right diagonal hatching;2;5;0;0
3;standard;Wide cross-hatching;2;6;0;0
4;standard;Wide diagonal cross-hatching;2;7;0;0
5;standard;Wide left diagonal hatching;2;8;0;0
6;standard;Wide horizontal hatching;2;9;0;0
7;standard;Wide vertical hatching;2;10;0;0
8;standard;Hollow;3;0;0
9;standard;Narrow right diagonal hatching;1;8;8
1;0;0;0;1;0;0;0
0;0;0;1;0;0;0;1
0;0;1;0;0;0;1;0
0;1;0;0;0;1;0;0
1;0;0;0;1;0;0;0
0;0;0;1;0;0;0;1
0;0;1;0;0;0;1;0
0;1;0;0;0;1;0;0
500;alpha;Translucent: alpha=32;32
501;alpha;Translucent: alpha=64;64
600;rop;ROP 1;1
601;rop;ROP 2;2
```

- Version 105 and newer versions of the file must be saved in the UTF-8 code page without a BOM (Byte Order Mark).
- Some index values are reserved. Please refer to the file for additional information.
- Comments appear prefixed by the `#` character and are ignored by MapLink.
- Include declarations allow for a more structured layout of the fill style file by segregating different categories of style into their own file.

The declaration is prefixed by the '`I`' character followed by the lower and upper ranges of the styles they contain.
The final semi-colon delimited value is filename relative to the current file. The format of this sub-file is the same as the root file and can in turn include other files. Should the file not be found at the location indicated, any associated `TSLPathList` will be checked instead.
One of the major benefits, also introduced, of using included sub-files is that these files can be delay-loaded or in other words only loaded when they are required.

- Section headings are a concept borrowed from the symbol lists and allows the categorisation of styles from the point of view of the run-time. They appear in the file prefixed with '`S`' followed by the section name, such as 'APP6A'.

Every style that appears after a section heading declaration will be associated with that section, although this does not include styles that appear in sub-files included within the section. Sub-files require their own section heading.
Section headings may appear more than once within the tree of files with all styles that appear under each of these section heading declarations being associated with the same section.
The SDKs now allow the querying of a style's associated section using the `getXxxStyleValue` methods.

- In order to make it easier for custom styles to be managed, a specific 'user' sub-file with a specific range of index values has been defined and appended to the end of the line styles file. This is called '`fillstyles\tslfillstylesuser.dat`'.

This file is not shipped by default with MapLink, but it's non-existence will not generate an error.
Custom fill styles should be added to this file, thus making it easier to manage user-defined styles across MapLink upgrades.

The remaining entries in the file begin with the fill style index (used with `TSLRenderingAttributeFillStyle`), the type (standard/alpha/rop) and a textual description that is displayed in MapLink Studio Feature Book. The rest of the fields for each entry are type dependant.

There are several different types of fill style:

- Standard Operating System: These are of various types indicated by the first custom field.

- Type 4: Solid. Subsequent fields for this entry are ignored.
- Type 3: Hollow. Subsequent fields for this entry are ignored.
- Type 2: Hatched. Next field is the hatch style. Subsequent fields for this entry are ignored.
- Style 5: Diagonal, 45 degree upward, left to right.
- Style 6: Horizontal and vertical crosshatch.
- Style 7: 45-degree crosshatch.
- Style 8: Diagonal, 45 degree downward, left to right.
- Style 9: Horizontal hatch.
- Style 10: Vertical hatch.
- Type 1: Patterned: Next two fields are width and height of the bitmap grid that follows. Each entry in the grid represents a pixel in the pattern. A 1 means that the pixel will be drawn in the current fill colour, a 0 means that the pixel will not be drawn – i.e. these pixels are transparent. MapLink has no concept of an opaque patterned fill.

  Note that patterned fills on certain platforms (Windows 98) are limited to 8x8. Any size that is not a multiple of 8 may incur a performance penalty.

  - Alpha blended: These are currently only available on Windows 2000 and newer platforms, along with X11 platforms that support the XRender extension. The custom field defines an alpha-blend in the range 0 to 255, where 0 is completely transparent and 255 is fully solid.
  - Raster Operation: These fill styles apply a raster operation code to the fill.
- R2_BLACK (1): Pixel is always 0.
- R2_COPYPEN (2): Pixel is the pen colour.
- R2_MASKNOTPEN (3): Pixel is a combination of the colours common to both the screen and the inverse of the pen.
- R2_MASKPEN (4): Pixel is a combination of the colours common to both the pen and the screen.
- R2_MASKPENNOT (5): Pixel is a combination of the colours common to both the pen and the inverse of the screen.
- R2_MERGENOTPEN (6): Pixel is a combination of the screen colour and the inverse of the pen colour.
- R2_MERGEPEN (7): Pixel is a combination of the pen colour and the screen colour.

- R2_MERGEPENNOT (8): Pixel is a combination of the pen colour and the inverse of the screen colour.
- R2_NOP (9): Pixel remains unchanged.
- R2_NOT (10): Pixel is the inverse of the screen colour.
- R2_NOTCOPYPEN (11): Pixel is the inverse of the pen colour.
- R2_NOTMASKPEN (12): Pixel is the inverse of the R2_MASKPEN colour.
- R2_NOTMERGEPEN (13): Pixel is the inverse of the R2_MERGEPEN colour.
- R2_NOTXORPEN (14): Pixel is the inverse of the R2_XORPEN colour.
- R2_WHITE (15): Pixel is always 1.
- R2_XORPEN (16): Pixel is a combination of the colours in the pen and in the screen, but not in both.

Note that ROP brushes are highly dependent for the effect on the underlying graphics engine implementation and some degree of experimentation may be necessary. Different graphics devices will interpret these in different ways – notably printers.

12.6.7 Symbols

The symbols file, usually called tslsymbols.dat, holds the definition of the visualisation for instances of `TSLSymbol` entities. The format is similar to the line styles file. The section names are used for display in MapLink Studio.

```
TSLSL 110                    // File ident and format version number
;                            // Field separator on subsequent lines
#This is a comment
I;2;14000;symbols/anothersymbolsfile.dat
S;Basic Shapes               // Section name for subsequent symbols
T;1;0;0;1;0;\MapLink 4.0\TMF\Circle Filled.tmf
S;UK Attractions (Icons)     // Section name for subsequent symbols
R;14001;16;16;\Attractions\DfT\Agricultural Museum.ico
S;MapLink 4.0 (Fixed Size)   // Section name for subsequent symbols
V;99000;0;0;0;1; Appears as SQUARE
  5;-3 -3; -3 3; 3 3; 3 -3; -3 -3;
S;Raster Symbols             // Section name for subsequent symbols
C;110000;16;16;1;\Rasters\Oil Well.png
S;Font Symbols              // Section name for subsequent symbols
F;120000;1
```

- Version 109 and newer versions of the file must be saved in the UTF-8 code page without a BOM (Byte Order Mark).
- Some index values are reserved. Please refer to the file for additional information.
- Comments appear prefixed by the `#` character and are ignored by MapLink.
- Include declarations allow for a more structured layout of the symbol file by segregating different categories of style into their own file. The declaration is prefixed by the '`I`' character followed by the lower and upper ranges of the styles they contain. The final semi-colon delimited value is filename relative to the current file.

The format of this sub-file is the same as the root file and can in turn include other files. Should the file not be found at the location indicated, any associated `TSLPathList` will be checked instead.
One of the major benefits, also introduced, of using included sub-files is that these files can be delay-loaded or in other words only loaded when they are required.

- Section headings allow the categorisation of styles from the point of view of the run-time. They appear in the file prefixed with '`S`' followed by the section name, such as 'APP6A'.

Every style that appears after a section heading declaration will be associated with that section, although this does not include styles that appear in sub-files included within the section. Sub-files require their own section heading.

Section headings may appear more than once within the tree of files with all styles that appear under each of these section heading declarations being associated with the same section.
The SDKs now allow the querying of a style's associated section using the `getXxxStyleValue` methods.

- In order to make it easier for custom styles to be managed, a specific 'user' sub-file with a specific range of index values has been defined and appended to the end of the symbol styles file. This is called `'symbols\tslsymbolsuser.dat'`. This file is not shipped by default with MapLink, but it's non-existence will not generate an error.

Custom symbols should be added to this file, thus making it easier to manage user-defined styles across MapLink upgrades.

There are five different types of symbol available. The first field for each entry line indicates the type; the second is the symbol ID used for setting the `TSLRenderingAttributeSymbolStyle` value.

Type T: These are TMF symbols, created in Symbol Studio. Subsequent fields after the symbol ID define the x and y origin of the symbol in the TMC space of the symbol itself, a scalable flag, the default rotatability of the symbol and the filename relative to the symbols file or in the `config/symbols` directory. Non-scalable TMF symbols are always drawn in the same TMC units as defined in the symbol and take no notice of the size defined on the symbol instance.

Type R: These are raster icon symbols. Subsequent fields after the symbol ID define the x and y origin of the symbol in the device unit space of the symbol itself, a scalable flag and the filename relative to the symbols file or in the `config/symbols` directory.

Type C: These are raster symbols. Subsequent fields after the symbol ID define the x and y origin of the symbol in the device unit space of the symbol itself, a scalable flag and the filename relative to the symbols file or in the `config/symbols` directory.

Type V: These are simple line vector symbols, always fixed size in device units. Subsequent fields after the symbol ID define the x and y origin of the symbol, a scalable flag (currently ignored) and the number of lines (N). After this there are N lines defined of the following form:

```
NumPointsInLine;x0 y0;x1 y1; … ;xN yN
```

Type F: These are font symbols which use a single character from a font as the symbol. The subsequent field after the symbol ID defines the font ID from the fonts file that the symbol will use.

12.6.8 Fonts

The fonts file, usually called tslfonts.dat, holds the definition of the visualisation for instances of `TSLText` entities. The format is similar to the line styles file.

```
TSLFNT 107                  // File ident and format version number
;                           // Field separator on subsequent lines
#This is a comment
I;3;55;symbols/anothersymbolsfile.dat
#Above is an include declaration to another file.
S;This is a section heading   // Section name for subsequent styles
1;0;Arial;100;0;0
2;0;Arial Black;100;0;0
56;1;TSLRom.thf
```

- Version 106 and newer versions of the file must be saved in the UTF-8 code page without a BOM (Byte Order Mark).
- Comments appear prefixed by the `#` character and are ignored by MapLink.
- Include declarations allow for a more structured layout of the fill style file by segregating different categories of style into their own file. The declaration is prefixed by the '`I`' character followed by the lower and upper ranges of the styles they contain. The final semi-colon delimited value is filename relative to the current file.

The format of this sub-file is the same as the root file and can in turn include other files. Should the file not be found at the location indicated, any associated `TSLPathList` will be checked instead.
One of the major benefits, also introduced, of using included sub-files is that these files can be delay-loaded or in other words only loaded when they are required.

- Section headings are a concept borrowed from the symbol lists and allows the categorisation of styles from the point of view of the run-time. They appear in the file prefixed with '`S`' followed by the section name, such as 'APP6A'.

Every style that appears after a section heading declaration will be associated with that section, although this does not include styles that appear in sub-files included within the section. Sub-files require their own section heading. Section headings may appear more than once within the tree of files with all styles that appear under each of these section heading declarations being associated with the same section.
The SDKs now allow the querying of a style's associated section using the `getXxxStyleValue` methods.

Of the remaining file entries, they appear with the following fields:

- Font ID used for `TSLRenderingAttributeTextFont`.
- Type : 0 = Operating System, 1 = Vector, 2 = Xft (see next section)
- Subsequent fields are operating system and font type dependent:

- For Windows, operating system fonts define the name, weight, italic and underline flags.
- For X11 platforms: The Type '0' is **deprecated** and should be avoided. Text drawn using the Type '0' font drawing will not display UTF-8 text correctly. You should migrate to Type '2'.
- For vector fonts, the rest of the line defines the name of a Hershey Font file used to render the scalable vector font. This type of font is very efficient for rendering rotated text but is a simple single pixel wide line. Vector fonts only support printable ASCII.

12.6.9 Xft Fonts (X11)

MapLink 7.0 and newer draws text using the Xft extension by default. This means that any font that is accessible through Fontconfig on the host system can be used.

MapLink 8.0 uses Pango and Xft for font rendering.

Strings rendered using the Xft extension can be drawn rotated. If you are using multiple threads for rendering, then you need to be aware that Xft is not thread safe. We have exposed two methods; `TSLMotifSurface::lockXft()` and `TSLMotifSurface::unlockXft()` when you draw text using the Xft extension.

It is possible to draw strings which are represented as UTF8. However, this is not officially supported as the layout engine is very simple. If you have a requirement to draw non-ASCII text please contact sales/support so that we can gauge the demand.

MapLink accepts Fontconfig pattern strings, allowing full control over font appearance. The following is an example from the 'X11' tslfonts.dat. The bold section shows several example patterns.

```
TSLFNT 107
;
1;2;Helvetica:weight=medium:slant=roman;0;0
12;2;Helvetica:weight=medium:slant=oblique:width=condensed;0;0
16;2;Bookman:weight=light:slant=italic;0;0
```

More information about the naming convention can be found here:

- http://www.freedesktop.org/software/fontconfig/fontconfig-user.html

The section 'Font Properties' is a list of valid properties. The 'Font Names' is the definition of the formatting of the strings.

Applications desiring the text rendering behaviour from MapLink 6.0 and earlier should replace the default `tslfonts.dat` fonts file with `tslunixbitmapfonts.dat` from the MapLink `config` directory.

12.7 APP-6A and 2525B Symbology

MapLink supports most APP-6A and 2525B symbology using the `TSLAPP6ASymbol` and `TSLAPP6AHelper` classes.

To choose between APP-6A or 2525B symbols, you need to load the appropriate configuration file thus:

```
string config( TSLUtilityFunctions::getMapLinkHome() );
config.append( "/config/app6aConfig.csv" ); // the default config file

TSLAPP6AHelper *symbolHelper = new TSLAPP6AHelper( config.c_str() );
if ( !symbolHelper->valid() )
  ...  // error
```

You can also choose to use either a filled set of symbols or unfilled set of symbols by loading the corresponding configuration file:

TSLAPP6AHelper configuration file	Description
<MAPLINK_HOME>\config\app6aConfig.csv	APP-6A, frames are filled
<MAPLINK_HOME>\config\app6aUnfilledConfig.csv	APP-6A, frames are not filled
<MAPLINK_HOME>\config\2525bConfig.csv	2525B, frames are filled
<MAPLINK_HOME>\config\2525bUnfilledConfig.csv	2525B, frames are not filled

The default constructor to `TSLAPP6AHelper` will load from:

- `<MAPLINK_HOME>\config\app6aConfig.csv`

You can choose to either obtain symbols as bitmaps (GDI and X11 only) or as a vector representation (`TSLEntitySet`). It is recommended to avoid the raster version on X11 due to X-Server resource constraints.

The vector representation is the only valid option for the OpenGL drawing surface.

```
const char fighterId[] = "1.x.2.1.1.2";

TSLAPP6ASymbol theSymbol;
if ( !symbolHelper->getSymbolFromID( fighterId, theSymbol ) )
  ...  // error

theSymbol.hostility( TSLAPP6ASymbol::HostilityHostile );
theSymbol.designation( "ABC123" );
theSymbol.heightType( TSLDimensionUnitsPixels );
theSymbol.height( 100 );
theSymbol.x( 400000000 );  // TMCs
theSymbol.y( 0 );

TSLEntitySet* es = symbolHelper->getSymbolAsEntitySet( &theSymbol );
if ( !es )
  ...  // error

// TSLStandardDataLayer* stdDataLayer;
stdDataLayer->entitySet()->insert( es ); // takes ownership of new set
stdDataLayer->notifyChanged( true );
// draw...
```

And we get something like this:

12.8 Raster Display

Many GIS based applications need to display information in raster format. Some sources of raster data, such as CADRG or ASRP charts, are likely to be embedded within a MapLink Studio map. These forms of raster image are typically processed through MapLink Studio and as such, would be displayed automatically when the map is loaded into a `TSLMapDataLayer`.

Other examples, such as aerial photography, bathymetric soundings or satellite images, are more likely to be displayed in addition to the map. These images may be displayed using the `TSLRasterDataLayer`. This is instantiated and added to a `TSLDrawingSurface`, just like any other Data Layer.

The `TSLRasterDataLayer` has no Coordinate System of its own and does no run-time projection of the raster image. It is therefore imperative that any raster added to a `TSLRasterDataLayer` must be in the same Coordinate System as the currently loaded map. For example, a side-scan sonar system outputs a raster in the local UTM zone. This raster could be added into a `TSLRasterDataLayer` correctly if it was being displayed on a Drawing Surface containing a map with the same UTM zone.

12.8.1 Adding Rasters

To add a raster image to the Data Layer, use the method `TSLRasterDataLayer::addRaster`. This method takes the unique name of the raster and the coordinates of the bottom left and top right corners of the raster in internal TMC units. An additional boolean flag allows control over whether the raster is pre-loaded or only in memory when it is being rendered. The unique name may be the full pathname of the raster file. Alternatively, the unique name may be the simple filename of the raster file, and a `TSLPathList` may be added to the Data Layer to indicate in which directory the file may be found.

12.8.2 Adding Masks

Many raster images added to a `TSLRasterDataLayer` will be projected in some way. Others may contain non-rectangular data or pixels that are unassigned. Such images may have an associated mask in order to hide the appropriate pixels - in effect, make them transparent. A mask is a 1-bit monochrome image that must be the same size as the associated raster image. When a mask is attached to a raster, the only pixels displayed are those whose corresponding mask pixel is set. MapLink Studio automatically creates masks when appropriate and places these alongside or embedded with, the raster image.

To add a mask to a raster, use

```
TSLRasterDataLayer::addMask( rasterName, maskName )
```

Again, the names may either be the full pathname or simple filename depending upon whether a `TSLPathList` has been attached to the Data Layer.

12.8.3 Raster Pyramids and Supported Formats

MapLink Studio, and Windows based applications can load raster images in many different formats - see the Deployment section for details of the associated dependencies. MapLink Studio currently outputs images in TIFF, PNG or JPEG format. All these formats can be read on Windows or X11 systems.

There is one other file format, the Envitia Raster Pyramid. This is a highly optimised form of raster image, which encapsulates reduced resolution images, tiling of high-resolution images and automatic embedding of associated masks. This format is typically output by MapLink Studio, but a simple flat raster may be converted to a Raster Pyramid using the `TSLRasterUtilities::rasterToPyramid` method. An associated method, `pyramidToRaster`, does the reverse conversion. The image data embedded within the Raster Pyramid is usually in TIFF, PNG or JPEG format.

12.9 Loading and Saving Data Layer Contents

Many applications need to display information from an external data source. Others need to make the information they display persistent. Several of the `TSLDataLayer` derivatives can load and/or save their contents. The exact capabilities vary between Data Layer types.

The `TSLMapDataLayer` can display a map which has been generated by MapLink Studio. The contents of the map are defined and referenced by a "`.map`" or "`.mpc`" file. To load the map into the `TSLDataLayer`, use the `TSLMapDataLayer::loadData` method.

The `TSLStandardDataLayer` can both load and save its contents, either via a file or via an in-memory buffer. To read from or write to a file, use the `loadData` and `saveData` methods. Each takes the filename to use. To read from or write to an in-memory buffer, use either `loadDataFromBuffer` or `saveDataToBuffer`. The `loadDataFromBuffer` method should be passed a buffer that the application has created, along with the size of the data that the buffer contains. The `saveDataToBuffer` method will create a buffer of the appropriate size, populate it with the contents of the Data Layer and return it to the application along with the size. This buffer may then be stored however the application requires, for example into a database. Once the application has finished with the buffer, it should call the `deleteBufferData` method so that the buffer can be destroyed. Note that the `loadDataWithConfig` and `saveDataWithConfig` methods will additionally persist the rendering and feature class list of the Standard Data layer.

The `TSLRasterDataLayer` is similar to the `TSLStandardDataLayer` in that it has the same load/save methods via a file or buffer. However, it does not necessarily store its contents directly. Instead, it can either store the meta-data of its contents or the meta-data and its contents. For each raster being displayed, the meta-data contains the filename, its referencing coordinates and the filename of any mask associated with the raster. The ability to save the raster images is only available when saving to a file, not to a buffer.

All of the `loadData` and `loadDataFromBuffer` methods will first clear the current contents of the Data Layer. The `TSLStandardDataLayer` also contains methods to append data to

the current contents without clearing them first. These are `appendData` and `appendDataFromBuffer`.

12.10 Interoperability

Interoperability in this context means the exchange of data between MapLink and other GIS formats. This is most obvious in MapLink Studio but is also available for a limited subset via the run-time Core SDK. The essential steps for importing or exporting are the same regardless of data format.

The steps for importing data are as follows

- Unlock the required interoperability support using `TSLUtilityFunctions::unlockSupport`.
- Create an array of `TSLFeatureMapping` objects, defining the native features to be imported, and the MapLink featureID that will be stored on them on import. This also defines the Render Level for objects of that feature type.
- Load the file into a `TSLStandardDataLayer` using the `TSLUtilityFunctions::importData`
- Create an instance of `TSLInteropImportSet` and add the imported layer to it.
- Continue this until all related files are added to the import set, in different layers.
- Create an instance of `TSLInteropManager` and pass the import set to the `postImportProcess` method. This method reconstructs any TMF complex primitives that have been encoded in the data – such as Bordered Polygons.
- The call to `postImportProcess` will merge the contents of the import set into a new Standard Data Layer and return it for use by the application.
- Destroy the import set and related Data Layers since they are no longer required.
- Note that more flexible formats may not require the use of an import set since they are heterogeneous.

The steps for exporting data are as follows

- Unlock the required interoperability support using `TSLUtilityFunctions::unlockSupport`.
- Ensure that all required data is in a single `TSLStandardDataLayer`.
- Create an instance of `TSLInteropManager` and pass the Standard Data Layer to the `preExportProcess` method. This method deconstructs any TMF complex primitives that exist in the Data Layer – such as Bordered Polygons.

- The call to `preExportProcess` will create an instance of a `TSLInteropExportSet` and return it to the application. This export set may contain multiple Standard Data Layers and filenames.
- Use `TSLUtilityFunctions::exportData` to export each Data Layer in the export set.
- Destroy the export set and related Data Layers since they are no longer required.

The processing that occurs in the `TSLInteropManager` methods is highly configurable. See the detailed online documentation of `TSLInteropConfig` for further details.

12.10.1 MapInfo MIF/MID Format

This format is a pair of simple ASCII files, one (MIF) file contains the geometry and meta-data; the other (MID) file contains the attributes for each geometric entity. Like TMF, MIF files can contain a mixture of entity types and it therefore provides a good match. Unless the data to be exchanged contains Bordered Polygons, it may not be necessary to process the data using the Interop Manager.

MIF files can contain rendering information. Since the MapLink and MapInfo styles are different, a mapping between the two sets of styles must be defined. The default mapping is in the "`<MAPLINK_HOME>\config\mifinteroperability.ini`" file, which should be passed to the `TSLUtilityFunctions::importData` method. Details of each mapping are defined in comments in the file.

12.10.2 OS MasterMap Format

This format is an XML file based upon GML 2.1.2. Like TMF, OS MasterMap files can contain a mixture of entity types. This format is mainly supported for the Seamless Layer Management of the MasterMap data. It usually does not require processing using the Interop Manager.

For change only update files, a departed feature is identified as a `TSLSymbol` instance with a featureID of (-255) and the topological ID (TOID) stored in the entityID.

MasterMap files can contain many attributes that may not be of interest. The set of attributes to be imported must be defined. The default mapping is in the "`<MAPLINK_HOME>\config\OSMasterMap.ini`" file, which should be passed to the `TSLUtilityFunctions::importData` method. Details of each entry are defined in comments in the file. The TOID, Feature Code, Version and Type attributes are always imported. Note that on Ordnance Survey advice, any primitives with the 'broken' attribute set true will be ignored.

- The TOID may be accessed using `TSLEntityBase::entityID`.
- The Feature Code may be accessed using `TSLEntityBase::featureID`.
- The Version may be accessed using `TSLEntityBase::version`.
- The Type attribute is used to determine which Entity type to create.

12.10.3 ShapeFile Format

This format is a complex binary format (.shp) file, with an associated database (.dbf) file. Every object in the file must be of the same entity type and so a typical TMF file, which is usually heterogeneous, will need processing using the Interop Manager.

12.10.4 OS NTF LandLine Format

This format is a simple ASCII file. The filter does not require a Feature Mapping array and does not import or export attributes. The featureID stored on the Entity is assumed to be the NTF LandLine feature code for import and export.

12.10.5 Attribute Information

Attribute data from directly imported vector data is held on the `TSLDataset` object on each entity.

Firstly, obtain the `TSLEntitySet` from the `TSLStandardDataLayer` that the source data was imported into. This should then be iterated through, checking the entity type, recursing, if necessary, i.e. if the entity is another `TSLEntitySet`.

Note: For the SHP data format, as it is a flat format, it should only contain entities of a single type, for example a shapefile rivers.shp would only contains lines, a shapefile countries.shp would only contains polygons, etc...

Retrieve the `TSLDataSet` object from the `TSLEntity`. The attribute data is held in `TSLVariants` accessed by keys and fields on the dataset.

At this point it is necessary to have some knowledge of the attributes that you're expecting to find. For example, the rivers.shp shapefile may contain the following attribute fields:

- RIVER_
- RIVER_ID
- NAME
- SYSTEM

Note: The shapefile specification does not say what attributes are present or should be present. The attributes could be different in each file.

From the `TSLDataSet` the number of available keys/fields can be determined and then the keys and/or fields accessed by index using the functions:

- `getAvailableKey`
- `getAvailableField`

Obviously, for the rivers.shp file above the number of available keys/fields returned would be four.

The keys and fields are returned as `TSLSimpleStrings`. So, if the contents of the NAME attribute are required, iterate through the `TSLDataSet` retrieving the Available Fields until a

field called NAME is returned. Then retrieve the actual data using one of the `getData` functions.

The following code snippet retrieves the Name attribute data (i.e. the actual name of the river, e.g. Amazon) and sets it into the entity name property:

```
// Entity Set off of StandardDataLayer containing imported shp file rivers.shp
if (TSLEntitySet* set = TSLEntitySet::isEntitySet( m_stdDataLayer->entitySet() )
{
  int setSize = set->size();
  for( int i(0); i < setSize; ++i )
  {
    // Get each entity in the set
    // Ideally the entity type should be checked for Entity Set
    // and recursed if found
    // In this code sample I know all entities are TSLPolylines
    if (  TSLEntity* entity = (*set)[i] )
    {
      // Retrieve the TSLDataSet from the entity
      const TSLDataSet* dataSet = entity->dataSet();
      if( dataSet )
      {
        // Iterate through the available fields
        // In the rivers.shp file there are four fields, RIVER_, RIVER_ID,
        // NAME & SYSTEM
        int dsetNumFields = dataSet->numAvailableFields();
        for( int j(0); j < dsetNumFields; ++j )
        {
          TSLSimpleString fieldName;
          dataSet->getAvailableField( j, fieldName );

          // In this code sample we are interested in the NAME field
          if ( fieldName == "NAME" )
          {
            // Extract the data from the field
            // In this case the name of the river as a string
            const TSLVariant *variant = dataSet->getData( fieldName.c_str() );
            if( variant )
            {
              int len = variant->getValueAsString( 0, 0, 0 ) ;
              char * buf = new char[ len + 1 ] ;
              variant->getValueAsString( buf, len ) ;

              // Do something with the data
              entity->name( buf );
            }
          }
        }
      }
    }
  }
}
```

12.11 Layer History Management

The core MapLink SDK allows data layers to maintain a version history. Currently, layer history is restricted to map data layers that contain Seamless Detail Layers. Using the 'publish and archive' features of the Seamless Layer Manager, it is possible to capture the history of changes applied to the map. It is then possible to rollback the map to any point

in time using the new data layer 'flashback' mechanism. It is also possible to query the data layer for the version history of any extent within the layer.

The following shows the typical steps required in order to create and maintain a map layer with history information (error handling is omitted for brevity):

```
TSLSeamlessLayerManager* slm = ...;

slm->enablePublishing( true, dir ); // Publishing must be enabled

slm->initialiseExistingLayerForIngest( mapPath, layerName );

TSLStandardDataLayer* layer = ...;
// Load the map data into 'layer'
TSLUtilityFunctions::importData( layer, ... );

// Import the layer into the seamless layer manager
slm->ingestData( layer );

// Finish the import process
slm->finalise();

TSLTimeType timestamp;
_time64( &timestamp ); // The timestamp to attach to this version

// Query the map's history
TSLVersionHistorySet const* history = map->versionHistory();

// Get the version number that will be used when the next archive
// is created.
TSLHistoryVersion version = history->getCurrentArchiveVersion();

// We store the archives in a hierarchical tree based on the
// version number e.g. <rootArchiveDir>\1\, <rootArchiveDir>\2\,
// <rootArchiveDir>\3\, etc. where <rootArchiveDir> is the root
// directory that contains the archives for the layer e.g.
// 'c:\MyMap\archives\LayerA\'.
char buffer[16] = { '\0' };
sprintf( buffer, "%d", version );
string archiveDir = rootArchiveDir;
archiveDir += buffer;

// The manager will automatically increment the archive version
// number that is to be used for the next archive
slm->publishAndArchive( archiveDir.c_str(), timestamp );

slm->enablePublishing( false ); // Finished publishing
```

12.12 Filter Data Layers

The MapLink Filter Data Layer provides a simpler interface than the interoperability described earlier in this section and is intended to be intuitive to users familiar with MapLink Studio. Currently the Filter Data Layer only supports two input formats; NITF and raster images. A user should create a Filter Data Layer for each data file that they intend to display at the same time. In MapLink Studio a user would organise their data files into

datasets and layers, but it is intended that mimicking the organisation of layers should be handled by the application.

The other main difference compared to MapLink Studio is the concept of 'Display Items' which were introduced to better support the loading of NITF data. A display item is a sub-object within the data file, be it a raster or vector item. When loading raster data, only one Display Item will exist, while a NITF file may contain hundreds.

The steps needed to use the basic functionality of the Filter Data Layer are as follows

- Create the required `TSLFilterDataLayer` derivative
- Unlock the data format, if required, using the `unlockSupport` function.
- Set the temporary directories used to store intermediate files using the `setDirectories` function if required.
- Load the data file
- Set the output coordinate system and/or linear transform on the layer if required
- Set the input coordinate system and/or geo-location on each display item. These can be retrieved by querying the layer using the `getDisplayItemAt` function.
- Add the layer to a drawing surface for display

The loaded data can be saved to file in a standard format by querying the Display Item for the internal MapLink Data Layer they represent. A Display Item containing raster data will return a `TSLRasterDataLayer` and a vector Display Item, a `TSLStandardDataLayer`.

The following is an example of using this layer:

```
// Load tsltransforms.dat this only needs to be done once at application
// startup.
TSLCoordinateSystem::loadCoordinateSystems();

// Create a Filter Data Layer which uses the GeoTIFF Filter
m_filterDataLayer = new TSLRasterFilterDataLayer(TSLFilterTypeGeoTIFF,
                                                 "GeoTIFFFilter");

// Raster processing options these are the same as you find in MapLink Studio
m_filterDataLayer->rasterSplitThreshold( 256 );
m_filterDataLayer->rasterPyramidOptions(TSLRasterInterpolationBilinear, false,
                                        TSLRasterTypePNG, 2);

// Tell the layer how many threads you want to use when re-projecting the
// raster.
m_filterDataLayer->setRasterThreadingOptions(0, 0);

// You can reduce the resultant image to 8 bit with 256 colours by
// uncommenting this line
//m_filterDataLayer->rasterOptions(24, 256);

// Set the Output coordinate transform
// This matches the Natural Earth World map that is shipped with MapLink.
const TSLCoordinateSystem *dynArcCS =
```

```
                            TSLCoordinateSystem::findByName( "Dynamic ARC Grid" );
    TSLCoordinateSystem *outputCS = dynArcCS->clone( 1000 );
    outputCS->setTMCperMU( 50.0 );
    m_filterDataLayer->setCoordinateSystem( outputCS );

    // Load the data.
    m_filterDataLayer->loadData(pathToGeoTiff.c_str());

    // Query the first display item - note there could be more than one.
    TSLFilterDataLayerDisplayItem *displayItem =
                            m_filterDataLayer->getDisplayItemAt( 0 );

    // You should set any additional processing options at this stage.
    //
    // This could be what the input coordinate system is or the geo-location.
    // In the case of the GeoTIFF filter this is not necessary.
    //
    // rasterItem->setInputCoordinateSystem( inputCS );
    // rasterItem->setGeolocation( 0.0, 0.0, 700000.0, 1300000.0 );

    // Process the layer... if you do not call this the layer will be processed on
    // first draw.
    m_filterDataLayer->process();

    // Save the processed data so we can load it rather than re-process.
    //   - you need to know if it is vector or raster on reload so that you can
    //     create either a TSLStandardDataLayer or TSLRasterDataLayer
    displayItem->saveDataLayer(locationToSaveProcessedDataTo.c_str(),
                            TSL_MAPLINK_DEFAULT_VERSION, TSLRasterTypePNG, 2);

    // Add the layer to a drawing surface
    m_surface->addDataLayer( m_filterDataLayer, "filterlayer");
```

The processing of large amounts of raster data can take some time, it is therefore recommended that the processing takes place in a background thread. If you take this approach do not add the layer to a drawing surface until after the processing has been completed. The layer must be owned by the thread which owns the drawing surface.

We would recommend that you only process one layer at a time in a background thread.

12.13 Web Map Service Data Layer

The MapLink Web Map Service Data Layer is a Data Layer that allows efficient loading of Open Geospatial Consortium (OGC) standardised Web Map Servers (WMS). The layer currently supports the loading of raster data from remote WMSs that implement version 1.1.1 of the standard, although future MapLink releases may support further versions.

Although setting up the layer is similar to the use of other MapLink Data Layers, due to the use of a multi-threaded file loader, issues surrounding thread safety may need to be taken into consideration. The MapLink C++ API tries to ensure this safety using const methods so that the user can modify variables that may break thread safety only when permitted. The .NET APIs follow similar rules but enforce them by returning failures from methods that may not be called if they break the thread safety. Refer to the API documentation for more information on these methods.

To setup the layer the user must provide an implementation of a callback class which is polled whenever the layer needs additional information. Each of the methods in this callback class provides sufficient parameters for the user to modify the layer details and will always occur in a thread safe manner. Settings should not be changed when the process is not currently executing one of the callback methods.

The layer also supports the use of layer dimensions and styles along with all the other service parameters supported by the WMS standard. The only part of the standard that is not supported by the MapLink WMS Data Layer is the use of the optional GetFeatureInfo request.

A good starting point when developing using the WMS Data Layer is to refer to the WMSClientSample supplied with MapLink. It demonstrates how to use the WMS in a thread safe manner and permits the setting of WMS dimension and style settings.

There are certain limitations with its use in conjunction with MapLink 3D drawing surfaces due to the coordinate system support.

13 OpenGL Drawing Surface

The OpenGL drawing surface allows an application to take advantage of hardware acceleration to enable high performance visualisations on both desktop and mobile platforms. In many circumstances it can be used as a drop-in replacement for the GDI-based and X11-based drawing surfaces from the Core SDK.

13.1 Library Usage and Configuration

As of version 11.1, MapLink is no longer supplied with Debug or 32-bit libraries. Therefore, your application's build should link against the Release Mode libraries in all configurations.

MapLinkOpenGLSurface64.lib
Release mode, DLL version.
Uses Multithreaded DLL C++ run-time library.

Must also link the MapLink CoreSDK library MapLink64.lib

Requires TTLDLL preprocessor directive.

Refer to the document "MapLink Pro X.Y: Deployment of End User Applications" for a list of run-time dependencies when redistributing. Where X.Y is the version of MapLink you are deploying.

13.2 Hardware Requirements

The OpenGL surface requires **OpenGL 2.1** or later (for desktop) or **OpenGL ES 2.0** (for mobile/embedded) compliant hardware in order to function. Systems that do not meet this requirement should use either the `TSLNTSurface` on Windows or the `TSLMotifSurface` on X11 platforms.

Note: **Integrated motherboard chipsets, such as Intel Integrated Graphics, should be avoided if at all possible for desktop/laptop applications.**

The table below lists any additional extensions that will be used by the surface depending on the OpenGL version it is running on. Recommended extensions are not required for the surface to operate but may have a negative impact on performance or functionality if not present.

OpenGL Version	Required Extensions	Recommended Extensions
2.1	EXT_framebuffer_object or ARB_framebuffer_object	NV_primitive_restart[1] ARB_texture_multisample[2]
3.2	None	EXT_direct_state_access[1]
ES 2.0	None	OES_standard_derivatives[3] OES_element_index_uint[4]

1. These extensions allow for improved performance if present but do not affect the functionality offered by the drawing surface.

2. Without this extension multisample anti-aliasing cannot be applied to buffered or transparent layers.
3. If this extension is not present very large pieces of text (> 150 pixels) may have noticeable aliasing.
4. If this extension is not present the surface is limited to rendering polygons or polylines with a maximum of 65536 coordinates. Any polygons or polylines with more than this number of coordinates will not display correctly. This is a hardware limitation.

13.3 Where to Begin?

The first question to ask is whether the OpenGL surface is suitable for your application. There are several points to consider when answering this question, including:

The availability of hardware acceleration on the target devices. An application intended to run primarily on virtual machines or very old hardware will likely not gain any benefit using the OpenGL surface over the `TSLNTSurface` or `TSLMotifSurface`.

Any custom rendering the application will perform. If the application contains a large amount of GDI or Xlib rendering code it will be more difficult to integrate the OpenGL surface into the application than the `TSLNTSurface` or `TSLMotifSurface`. Additionally, the overhead of merging the output of the two separate rendering APIs may outweigh any performance improvements gained from using hardware acceleration.

Any requirements on identical rendering output on different hardware. OpenGL does not require implementations to produce identical outputs from the same set of rendering commands, so the same application will produce slightly different output when run on different hardware. These differences are not usually visible to the eye but will show up when performing a binary comparison between the output of different systems.

Developer familiarity with OpenGL. The drawing surface is not a complete graphics engine; therefore, some knowledge of OpenGL is necessary to implement any custom drawing required beyond that offered by the MapLink rendering API (`TSLRenderingInterface` and geometry).

13.3.1 Graphics Drivers

When using the OpenGL drawing surface, it is vitally important to use up-to-date graphics drivers for your hardware. Old graphics drivers can have missing features, poorer performance and bugs that can cause the drawing surface to malfunction or render incorrectly. Upgrading to the newest graphics drivers for your hardware should always be first step in diagnosing problems.

13.3.2 Which Class Should be Used?

The OpenGL surface is accessed through a variety of window system interface classes, similar to how `TSLNTSurface` provides an interface between MapLink and Windows-based systems and `TSLMotifSurface` provides and interface between MapLink and X11-based systems.

The interface classes for the OpenGL surface are named based on the interface they use as follows:

Windowing System	Applicable OpenGL Surface Classes
Windows	TSLWGLSurface
X11	TSLGLXSurface
Embedded/Mobile	TSLEGLSurface TSLNativeEGLSurface[1]

1. This class may not be present on all platforms - see the Release Notes for your platform for more information.

The `TSLOpenGLSurface` class contains the common functionality across all platforms.

13.3.3 What is the Difference Between TSLEGLSurface and TSLNativeEGLSurface?

On some embedded platforms there is a choice between these two window system interface classes. Both are intended to be used with OpenGL ES 2.0 systems via EGL, however `TSLEGLSurface` does not link against the system EGL library and thus cannot create or manage OpenGL contexts itself. This makes it usable across multiple different systems with different EGL libraries, whereas `TSLNativeEGLSurface` is tied to a specific EGL implementation.

Generally, you should use `TSLNativeEGLSurface` when:

- You want the easiest way of creating a MapLink drawing surface.
- You are not integrating with other code that performs its own OpenGL context management.

You should use `TSLEGLSurface` when:

- `TSLNativeEGLSurface` is not available.
- You wish to manage all OpenGL contexts yourself.

Unlike the other window system interface classes, `TSLEGLSurface` only has one constructor that takes no arguments. This creates the drawing surface in a detached state (where it is not associated with an OpenGL context), so the attach method must be called from a thread that has the OpenGL context to use bound to it. For the `TSLEGLSurface` the `requiresDisplayMetrics` method will always return true, so the

application must call `setDeviceCapabilities` or `setDisplayMetrics` with the appropriate values before any drawing is performed.

13.3.4 Additional Data Layers for use with the OpenGL Surface

When using the OpenGL drawing surface to display maps created by MapLink Studio, applications have a choice of two data layers, the `TSLMapDataLayer` and the `TSLStaticMapDataLayer`. Each layer offers a different trade-off between features and performance, so the best choice for an application depends on how the layer will be used. Section 13.6 covers the differences between these layers and when an application would want to use each one.

13.4 Realtime Reprojection

The OpenGL Drawing Surface supports the concept of reprojecting vector and raster at runtime on the GPU allowing for the projection centre to change for every frame.

The following projections are currently supported:

- Mercator
- Transverse Mercator
- Stereographic
- Gnomonic

Additional projections can be added if necessary, please contact sales@envitia.com to discuss.

OpenGL 3.3 or later is required with the following OpenGL extensions:

- `ARB_transform_feedback2`
- `ARB_shading_language_420pack`
- `ARB_draw_indirect`
- `ARB_gpu_shader_fp64 (FP64)`
- `ARB_shader_subroutine`

There are fallback mechanisms for all the above extensions, however the fallbacks have an impact on performance and complexity of the shaders.

The FP64 extension is one of the more critical extensions. Only a small number of the projections are supported if this extension is not present as we must emulate the 64bit double maths. This significantly complicates the shader code.

The following extensions are used if present:

- `ARB_texture_storage`
- `ARB_shader_image_load_store`

For additional information please refer to the MapLink Pro API documentation for the class TSLOpenGLSurface.

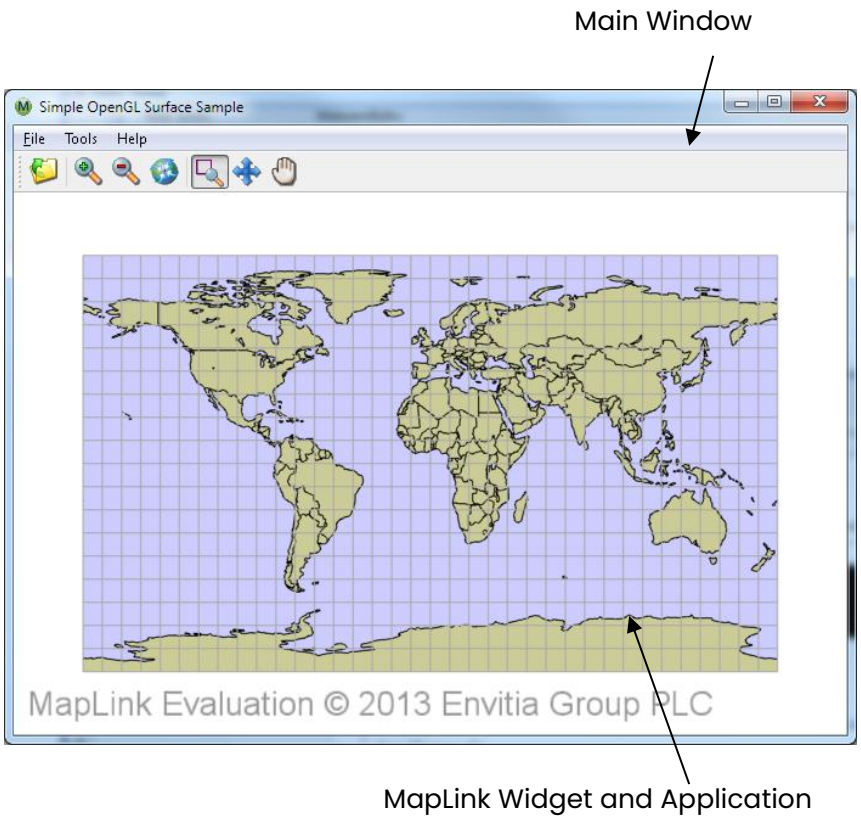
13.5 Walkthrough - The Simple OpenGL Surface Sample

A simple sample application that uses the OpenGL drawing surface is provided as part of a MapLink installation in order to demonstrate the basics of using the OpenGL drawing surface. The source code for this sample can be found in the `samples/Qt/SimpleGLSurfaceSample` folder of the installation. This sample uses Qt 5.0. There is also an MFC based sample based on the Simple Interaction sample to demonstrate the minimal changes necessary to convert to the new Drawing Surface.

The source files for the sample have the following purposes:

- **main.cpp** contains the entry point for the application.
- **mainwindow.cpp** and **.h** contains event handlers for toolbar and menu events, and is the class for the application's window.
- **maplinkwidget.cpp** and **.h** contain a simple custom Qt widget that the MapLink drawing surface is attached to. This class handles interactions with Qt events and callbacks, but uses the Application class below to interact with the MapLink API.
- **application.cpp** and **.h** contain the code that interacts with the MapLink API. This class is used by the widget above, but is split out into a seperate class in order to make it easier to follow the interactions with the MapLink API independently of interactions with Qt.

Visually, these items correspond to the following parts of the application:



13.5.1 Starting the Application - Choosing a Framebuffer Configuration

For convenience, the sample uses Qt to create the OpenGL context that will be used by the drawing surface. Before this context is created, we want to suggest to Qt that it uses a framebuffer configuration with certain parameters. This happens inside `main` in **main.cpp** with the following code:

```
// Qt will be creating the OpenGL context for us. In order for the
// drawing surface to work at its best we ask it to choose a
// framebuffer configuration with a specific set of
// parameters.
QGLFormat f;

// Request a 24-bit depth buffer. A 16-bit depth buffer will also work.
f.setDepthBufferSize( 24 );

// Request a double-buffered configuration to eliminate flickering when
// moving around the map
f.setDoubleBuffer( true );

// Request 4x multisampling anti-aliasing if supported by the hardware
f.setSamples( 4 );

// Request an OpenGL 3.2 core profile context if supported by the hardware
f.setVersion( 3, 2 );
f.setProfile( QGLFormat::CoreProfile );

QGLFormat::setDefaultFormat( f );
```

Once we have done this, the main window for the application can be created and displayed.

If we were letting the drawing surface create its own OpenGL context then the above would not be needed as the drawing surface would internally select a suitable framebuffer configuration.

13.5.2 Initialisation

When the main window of the application is created, this creates the custom widget (the MapLink widget from **maplinkwidget.cpp**), which in turn creates the application class (from **application.cpp**). These objects get constructed in this order:

1. MainWindow
2. MapLinkWidget
3. Application

The `MainWindow` constructor only deals with installing event handlers for the toolbar button and menu entries - it doesn't contain any interactions with MapLink.

Similarly, the majority of the `MapLinkWidget` constructor deals with ensuring event messages are propagated correctly to the widget. Additionally, it also instructs Qt to not

clear the drawing area automatically before any drawing occurs as we use the MapLink drawing surface to do this.

The constructor of the `Application` class tells MapLink to load its configuration files from an installation on the host machine by calling `TSLDrawingSurface::loadStandardConfig`. This must be done before any MapLink functionality can be used - without it maps cannot be drawn at all. This setup only needs to be done once in the application - usually at start-up.

13.5.3 Creating the Drawing Surface

The next step is the creation of the drawing surface. Qt will call `MapLinkWidget::initialiseGL` inside **maplinkwidget.cpp** to ask the widget to perform any initial setup. The MapLink drawing surface doesn't interface with Qt directly, it uses the underlying window handles from the operating system in order to allow it to be used with different application toolkits. Therefore, a small amount of platform specific code is needed in order extract these handles from Qt:

```
  // Platform Specific Setup.
#ifdef X11_BUILD
# if QT_VERSION < 0x50100
  // Extract the X11 information - QX11Info was removed in Qt5
  QPlatformNativeInterface *native =
                                  QGuiApplication::platformNativeInterface();
  Display *display = static_cast<Display*>( native->nativeResourceForWindow(
                                                        "display", NULL ) );
  Screen *screen = DefaultScreenOfDisplay(display);
#else
  // Qt 5.1 introduced a different version of QX11Info for accessing widget native
  // handles
  int screenNum = DefaultScreen( QX11Info::display() );
  Screen *screen = ScreenOfDisplay( QX11Info::display(), screenNum );
#endif
  // pass to the application as we will need for the Drawing Surface
  m_application->drawingInfo(display, screen);
#else
  // The MapLink OpenGL drawing surface needs to know the window handle to
  // attach to - query this from Qt
  WId hWnd = winId();

  // Pass the handle to the application so it can be used by the drawing
  // surface
  m_application->drawingInfo( hWnd );
#endif
```

These window handles are passed to the `Application` class for it to use when creating the drawing surface. The widget then calls `Application::create` in **application.cpp** to create the MapLink drawing surface and attach it to the window handles it just queried. Creating the drawing surface requires another small amount of platform specific code - the exact type of drawing surface used depends on the platform the sample is being run on:

```
  // Tell the drawing surface whether it will need to perform buffer swaps,
  // or whether it is handled externally. See the constructor of
  // MapLinkWidget.
  TSLOpenGLSurfaceCreationParameters creationOptions;
  creationOptions.swapBuffersManually( ML_QT_BUFFER_SWAP );

#ifdef X11_BUILD
  // Get the active OpenGL context to attach the drawing surface to
  GLXContext context = glXGetCurrentContext();
  GLXDrawable drawable = glXGetCurrentDrawable();

  // Create the Accelerated Surface object
  m_drawingSurface = new TSLGLXSurface( m_display, m_screen, drawable,
                                        context, creationOptions );
#else
  HGLRC context = wglGetCurrentContext();
  m_drawingSurface = new TSLWGLSurface( (HWND)m_window, false, context,
                                        creationOptions );
#endif

  if( !m_drawingSurface->context() )
  {
   // Error handling code
  }
```

In both cases the application queries the active OpenGL context created by Qt and instantiates a MapLink drawing surface, telling it to attach itself to this context.

The `TSLOpenGLSurfaceCreationParameters` class can be used to control the behaviour of the MapLink drawing surface - in this case it's used to tell the drawing surface not to perform buffer swaps after a draw as we have already told Qt that it should handle this.

Once the drawing surface is created, we check to make sure that the `context` method returns a valid value - if this returns `NULL` then the drawing surface did not attach successfully to the OpenGL context and cannot be used. The most common reason that this might occur is if the OpenGL implementation on the system does not meet the minimum requirements for the drawing surface.

Next, the application creates a data layer to load a map created by MapLink Studio into and adds this to the drawing surface so that its contents will be drawn when the drawing surface renders.

```
  // Add a map data layer to the drawing surface
  m_mapDataLayer = new TSLMapDataLayer();
  // Set a cache size of 256Mb on the map layer to avoid reloading tiles
  // from disk too frequently
  m_mapDataLayer->cacheSize( 256 * 1024 );
  m_drawingSurface->addDataLayer( m_mapDataLayer, m_mapLayerName );
```

At this stage the data layer is still empty, so nothing will be drawn yet.

The final step is the creation of the MapLink interaction mode manager and interaction modes.

```
// Now create and initialse the mode manager and modes
m_modeManager = new TSLInteractionModeManagerGeneric( this,
                                                      m_drawingSurface );

// Add the three interaction mode types to the manager - the zoom mode is
// the default
m_modeManager->addMode( new TSLInteractionModeZoom( ID_TOOLS_ZOOM ),
                        true ) ;
m_modeManager->addMode( new TSLInteractionModePan( ID_TOOLS_PAN ),
                        false ) ;
m_modeManager->addMode( new TSLInteractionModeGrab( ID_TOOLS_GRAB ),
                       false ) ;

// Display any errors that have occurred
const char *errorMsg = TSLErrorStack::errorString();
if( errorMsg )
{
  // Error handling code
}
```

The interaction modes are a set of premade event handlers that implement some common methods of manipulating the view of a drawing surface based on user input. In this case we set the manager up to have zoom to rectangle, pan to point and grab and drag modes. The zoom to rectangle mode is set to be the default active interaction mode.

13.5.4 Handling Window Resizing

Although attached to the widget, the drawing surface is still not in a valid state for drawing as it has yet to be told how big the window to which it is attached is.

After `MapLinkWidget::initializeGL` has finished, Qt will immediately call `MapLinkWidget::resizeGL` in **maplinkwidget.cpp**. This call is forwarded to `Application::resize` in **application.cpp**, which updates both the drawing surface and the interaction mode manager with the size of the window they are drawing to.

```
if( m_drawingSurface )
{
  // Inform the drawing surface of the new window size,
  // attempting to keep the top left corner the same.
  // Do not ask for an automatic redraw since we will get a call to
  // redraw() to do so
  m_drawingSurface->wndResize( 0, 0, width, height, false,
                               TSLResizeActionMaintainTopLeft ) ;
}
if( m_modeManager )
{
  m_modeManager->onSize( width, height ) ;
}
```

Now that the drawing surface knows the window size, it is fully initialised and ready for drawing.
This same event sequence is used to handle changes to the window size while the application is running, as the drawing surface must be notified of any changes to the window to which it is attached.

13.5.5 Drawing to the Window

When the sample's window needs to be redrawn, Qt will call `MapLinkWidget::paintGL` in **maplinkwidget.cpp**. This in turn calls `Application::redraw` in **application.cpp** to make the drawing surface draw all of the data layers inside it.

```
if( m_drawingSurface )
{
  // Draw the map to the widget
  m_drawingSurface->drawDU( 0, 0, m_widgetWidth, m_widgetHeight, true );

  // Don't forget to draw any echo rectangle that may be active.
  if ( m_modeManager )
  {
    m_modeManager->onDraw( 0, 0, m_widgetWidth, m_widgetHeight );
  }
}
```

There is one data layer in the drawing surface - the `TSLMapDataLayer` we added inside the `Application::create` method. Currently this has no map loaded, so drawing will just clear the widget to the default background colour, which is white.

13.5.6 Loading a Map

When the Open Map toolbar button or menu item is selected, Qt invokes the registered event handler for this action, which is `MainWindow::loadMap` in **mainwindow.cpp**. This displays the operating system's standard open dialog to let the user choose the map to load, and calls `MapLinkWidget::loadMap` in **maplinkwidget.cpp** to do the actual task of loading.

The widget does three things when loading a map:

1. Instructs the application class to load the map into its `TSLMapDataLayer`.
2. Tells the application to change the view of the drawing surface to cover the entirety of the new map, as the previous view of the drawing surface is unlikely to provide a useful view of the new map.
3. Asks Qt to redraw the window so that the new map can be seen, triggering the same call sequence as described in section 13.5.5.

The task of actually loading the data into the `TSLMapDataLayer` is done inside `Application::loadMap` in **application.cpp**.

```
if( !m_mapDataLayer->loadData( mapFilename.c_str() ) )
{
  QMessageBox::critical( m_parentWidget, "Failed to load map",
                         mapFilename.c_str() );
  return false;
}

if( m_modeManager )
{
  // Loading a map invalidates any stored views in mode manager  - this
  // sample doesn't create any
  m_modeManager->resetViews();
}
```

Changing the drawing surface view to cover the extent of the newly loaded map is done inside `Application::resetView`.

```
if( m_drawingSurface )
{
  // Reset the drawing surface rotation as well
  m_surfaceRotation = 0.0;
  m_drawingSurface->rotate( m_surfaceRotation );

  m_drawingSurface->reset( false );
}
```

The sample explicitly asks the drawing surface not to perform an immediate redraw as this is handled by `MapLinkWidget`. Any rotation that has been applied to the drawing surface by the sample is also removed.

13.5.7 Changing the View of the Map

Keyboard and mouse events that occur when the sample has focus are received by the `MapLinkWidget` in the `*Event` methods inside **maplinkwidget.cpp**. Each of these events is passed onto an equivalent event handler function inside the `Application` class in **application.cpp**. In turn, the `Application` class forwards these events to the MapLink interaction mode manager associated with the drawing surface. The manager will change the current view of the map based on the active interaction mode.

The return value of the interaction mode manager indicates to the `Application` class if a redraw is required based on the actions that have been taken. The `Application` passes this flag back to the `MapLinkWidget` as the return value from the event handler functions, which lets the `MapLinkWidget` trigger a redraw of the window when needed.

13.5.8 Changing the Active Interaction Mode

When the sample is started, the zoom to rectangle mode is the default interaction mode. This can be changed through the toolbar buttons or corresponding menu items, which invoke the registered action handlers in the `MainWindow` These actions are forwarded on to the `MapLinkWidget`, which in turn forwards them on to the `Application`.

The `Application` instructs the interaction mode manager to change the interaction mode using the unique identifiers assigned to each the interaction modes when they were initially added to the interaction mode manager when it was created.

```
void Application::activatePanMode()
{
  // Activate the pan interaction mode
  if( m_modeManager )
  {
    m_modeManager->setCurrentMode( ID_TOOLS_PAN ) ;
  }
}

void Application::activateZoomMode()
{
  // Activate the zoom interaction mode
  if( m_modeManager )
  {
    m_modeManager->setCurrentMode( ID_TOOLS_ZOOM ) ;
  }
}

void Application::activateGrabMode()
{
  // Activate the grab interaction mode
  if( m_modeManager )
  {
    m_modeManager->setCurrentMode( ID_TOOLS_GRAB ) ;
  }
}
```

13.6 Additional Data Layers for the OpenGL Surface

When using the OpenGL drawing surface applications can use an additional data layer called the `TSLStaticMapDataLayer`. This data layer is an alternative to the `TSLMapDataLayer` that does not support the more advanced features of the `TSLMapDataLayer`, but in return offers significantly higher drawing performance.

The `TSLStaticMapDataLayer` lacks the following features of the `TSLMapDataLayer`:

- Runtime projection
- Flashback
- Support for dynamic renderers
- Persistent tile caches

If any of these features are required in an application then the `TSLMapDataLayer` should be used, otherwise the `TSLStaticMapDataLayer` may be more suitable.

The API of the `TSLStaticMapDataLayer` is almost identical to that of the `TSLMapDataLayer` so it is simple for an application to switch them as desired. The primary difference between the two data layers is in the methods that control each layer's cache (`cacheSize` and `clearCache`). In the `TSLStaticMapDataLayer` this is split into two different sets of

methods, one used for drawing and one used for querying (such as the `findEntity` or `query` methods).

In addition to allowing for finer control over the memory use of the layer, the split allows for more flexible use of the data layer. Unlike the `TSLMapDataLayer`, with the `TSLStaticMapDataLayer` an application may retain the results of a query operation while the layer is redrawn as the separate caches mean changes to the drawing cache do not affect the objects returned by a query. It is also possible to use the data layer query methods on the `TSLStaticMapDataLayer` from a different thread to the one that the layer is drawn in, provided the application takes care to ensure that methods that would trigger a clearing of both caches (e.g. `loadData`) are not called without synchronisation inside the application. Issuing queries from more than one thread against the same data layer is not supported.

13.7 The Drawing Surface Coordinate System and Custom Data Layers

Separate from the coordinate system used for coordinate transformations in the drawing surface and data layers via a `TSLCoordinateSystem` object, the OpenGL drawing surface defines a different coordinate system that is used to map the OpenGL coordinate space to the screen. When all rendering is performed through MapLink data layers and the `TSLRenderingInterface` an application doesn't need to understand this coordinate space - it is all taken care of internally. However, when the application performs its own OpenGL rendering via a `TSLCustomDataLayer` or after MapLink has finished drawing it is important to understand how to correctly position items relative to the map.

The OpenGL drawing surface defines this rendering coordinate space to be the TMC extent of the area being rendered, without any drawing surface rotation or dynamic arc scaling included, centred on 0,0 (centre of the screen). The TMC extent that this equates to from the active `TSLCoordinateSystem` can be determined if required as follows, although it is generally not needed:

```
TSLDrawingSurface *surface = ...

TSLEnvelope renderExtent;
surface->getTMCExtent(renderExtent);

if(surface->getOption(TSLOptionDynamicArcSupportEnabled))
{
  // Dynamic arc is enabled, remove any scaling effect applied
  // to the envelope
  double tmcPerDUX = 0.0, tmcPerDUY = 0.0;
  surface->TMCperDU(tmcPerDUX, tmcPerDUY);
  renderExtent.scale(tmcPerDUY / tmcPerDUX, 1.0);
}

// renderExtent now contains the TMC extent of the surface's drawing
// coordinate system in the active TSLCoordinateSystem of the drawing
// surface
```

As mentioned above, this extent is centred on 0,0 when mapped to the drawing surface's rendering coordinate system (i.e., the bottom left of the envelope is equal to `-width()/2, -height/2` and the top right of the envelope is equal to `width()/2. height()/2)`. The bottom left and top right of this envelope map to the bottom left and top right of the screen - therefore 0,0 in the rendering coordinate system always maps to the centre of the screen.

13.7.1 Positioning Items in Practice

Like all OpenGL applications, the OpenGL drawing surface uses matrices to describe the transformations to perform when rendering items to the screen. These are divided into two matrices - the modelview matrix which describes any transformations to be applied within the rendering coordinate system, and the projection matrix which maps the results of this to a form OpenGL can interpret.

The drawing surface provides access to its matrices for the current or previous draw via the `modelViewMatrix` and `projectionMatrix` methods on `TSLOpenGLDrawingSurface`. The modelview matrix will already contain any drawing surface rotation or dynamic arc scaling that apply for the draw. Additionally, for convenience it also provides access to the TMC position that maps to 0,0 in rendering space via the `coordinateCentreX` and `coordinateCentreY` methods. Note that the centre coordinates are returned as doubles - they may not necessarily map to integer coordinate space.

The combination of these items allows for the construction of a modelview matrix that will correctly locate an item being drawn relative to the map. Generally, object drawn via OpenGL are object centred - i.e. their coordinates are defined relative to an origin local to that object rather than relative to the active coordinate system. Since coordinates in OpenGL must be provided in 32bit floats in most cases this is particularly important due to precision - the MapLink TMC coordinate space uses 32bits of integer precision but 32bit floats only provide 24bits of integer precision. By Object-centring and translating to the

correct location during a draw the problem of jittery movement of objects when operating near the edges of MapLink's TMC space does not occur.

Putting this all together, the code for positioning an object centred item for drawing inside a `TSLCustomDataLayer` might look like the following:

```
bool MyCustomLayer::drawLayer(TSLRenderingInterface *renderingInterface,
                              const TSLEnvelope* extent,
                              TSLCustomDataLayerHandler& layerHandler)
{
  TSLOpenGLSurface *surface = reinterpret_cast<TSLOpenGLSurface*>(
                                    layerHandler.drawingSurface());

  // This contains the TMC location of the centre of the object
  // in the drawing surface's active coordinate system
  TSLCoord objectTMCPosition = calculateObjectPosition();

  // Determine the modelview matrix to position the object
  // in the right place relative to the current drawing.
  Matrix modelViewMat(surface->modelViewMatrix());
  modelViewMat.translate(objectTMCPosition.x() -
                         surface->coordinateCentreX(),
                         objectTMCPosition.y() -
                         surface->coordinateCentreY());

  if(removeDynamicArcScaling)
  {
    // This will remove the effect of dynamic arc (if active)
    // from any subsequent transformations.
    double tmcPerDUX = 0.0, tmcPerDUY = 0.0;
    surface->TMCperDU(tmcPerDUX, tmcPerDUY);
    modelViewMat.scale(tmcPerDUX / tmcPerDUY, 1.0);
  }

  // Upload the matrices to OpenGL
  surface->stateTracker()->useProgram(m_program);
  glUniformMatrix4fv( m_modelViewUniform, 1, GL_FALSE,
                      modelViewMat.matrix());
  glUniformMatrix4fv( m_projectionUniform, 1, GL_FALSE,
                      surface->projectionMatrix());

  // Now draw the object
  ...

  return true;
}
```

13.7.2 Interspersing Custom Rendering with MapLink Rendering

The OpenGL drawing surface will internally change the order or defer drawing for performance reasons, using OpenGL's depth buffer to ensure items appear in the correct order on screen. When all rendering inside a custom data layer is performed by MapLink or implemented by the application there is no problem, and everything will work as expected. However, when mixing both MapLink rendering and custom rendering within the same data layer application drawn items may not necessarily appear in the expected order relative to the items drawn through MapLink.

In the simple case where all of the application rendering in the data layer occurs after all MapLink rendering, the application can use the `flushPendingDraws` method on `TSLOpenGLSurface` to ensure all MapLink draw commands have been sent to the GPU command stream before beginning application rendering. This might look as follows:

```
bool MyCustomLayer::drawLayer(TSLRenderingInterface *renderingInterface,
                              const TSLEnvelope* extent,
                              TSLCustomDataLayerHandler& layerHandler)
{
  TSLOpenGLSurface *surface = reinterpret_cast<TSLOpenGLSurface*>(
                                    layerHandler.drawingSurface());

  // This function performs rendering through the rendering interface
  doMapLinkDrawing(renderingInterface);

  // Ensure all MapLink rendering is done before we continue
  surface->flushPendingDraws();

  // Custom application rendering in this case does not use the
  // depth buffer
  surface->stateTracker()->disableDepthTest();

  // Custom application rendering occurs in here
  doMyCustomRendering();

  surface->stateTracker()->enableDepthTest();

  return true;
}
```

If the custom rendering is mixed in with MapLink rendering, then it can often be beneficial to make use of the depth buffer to ensure ordering in the same way that the drawing surface does. To assist in this the `TSLOpenGLSurface` provides the `acquireDepthSlice` method which can be used to reserve one or more depth buffer values for custom rendering use. The value returned is the depth in OpenGL's normalised device coordinate space, which since the OpenGL drawing surface uses an orthographic 2D projection and so has no perspective division is also OpenGL's clip space. The value can therefore be assigned directly to either `gl_FragDepth` in the application's fragment shader or to `gl_Position.z` in the application's vertex shader.

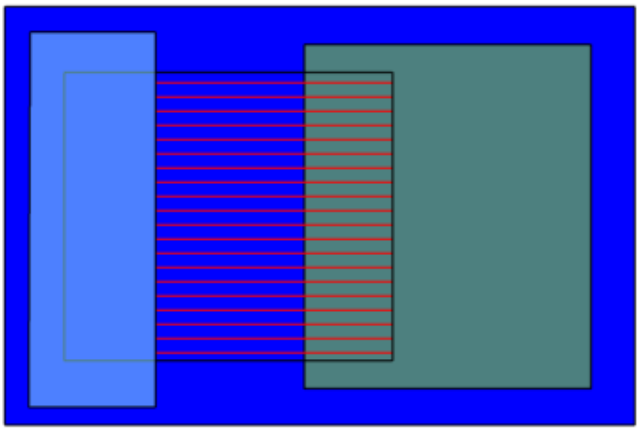
In the following examples `depthValue` is the value obtained from calling `acquireDepthSlice`:

```
// Vertex shader example - this method can be used on OpenGL ES 2.0 systems
// where gl_FragDepth is not available
#version 150 core

uniform float depthValue;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

in vec2 vertexPosition;

void main()
{
  gl_Position = (projectionMatrix * modelViewMatrix) * vec4( vertexPosition,
                0.0, 1.0 );
  gl_Position.z = depthValue;
}


// Fragment shader example
#version 150 core

uniform vec4 colour;
uniform float depthValue;

out vec4 pixelColour;

void main()
{
  gl_FragDepth = depthValue;
  pixelColour = colour;
}
```

When using the depth buffer in this fashion it is not necessary to call `flushPendingDraws` in order to ensure correct ordering. Depth buffer values obtained in this fashion only apply to the current data layer being drawn as the depth buffer will be cleared when drawing each data layer.

13.8 Transparency

In addition to per-layer transparency, the OpenGL drawing surface also allows for per-entity transparency via the `TSLRenderingAttributeEdgeOpacity`, `TSLRenderingAttributeExteriorEdgeOpacity`, `TSLRenderingAttributeFillOpacity`, `TSLRenderingAttributeTextOpacity` and `TSLRenderingAttributeSymbolOpacity` rendering attributes.
When using per-entity transparency, correct visualisation requires that strict back-to-front rendering order is used, however for performance reasons MapLink may internally rearrange the order items are drawn in. This may sometimes lead to the following situation:



This image contains four rectangles in the following order:

1. An opaque blue rectangle with black opaque edges.
2. A partially transparent green rectangle with black opaque edges.
3. A red rectangle with a patterned fill style with black opaque edges.
4. A partially transparent cyan rectangle with black opaque edges.

With this ordering, the patterned red rectangle should be visible through the partially transparent cyan rectangle, however after reordering the rectangles were drawn in this order:

1. The black opaque edges for all four rectangles.
2. The opaque blue rectangle fill.
3. The partially transparent green rectangle fill.
4. The partially transparent cyan rectangle fill.
5. The patterned red rectangle fill.

When drawn in this order, the section of the patterned red rectangle that should be visible through the partially transparent cyan rectangle has not been drawn at the point the cyan rectangle is rendered, so this rectangle's colour was blended with the opaque blue rectangle instead of the patterned red rectangle. Effectively it was drawn as if the patterned red rectangle did not exist. When the patterned red rectangle is drawn at the end, the section covered by the partially transparent cyan rectangle does not get drawn again as it is considered to already be obscured by an object in front (the cyan rectangle).

MapLink attempts to minimise the situations where this might occur. Specifically, it can only occur when the following conditions are true:

- Rendering that occurs within the same MapLink data layer.
- For data layers that contain multiple tiles (e.g. a `TSLMapDataLayer`), rendering that occurs within the same tile.
- Rendering that occurs at the same render level.
- Rendering that occurs at the same rendering pass (for multi-pass linestyles).

- The overlapping transparent items use different fill or line styles, or are different types of rendering (e.g. a polygon's fill and a polygon's edge).
- The item being drawn is not a `TSLSymbol`.

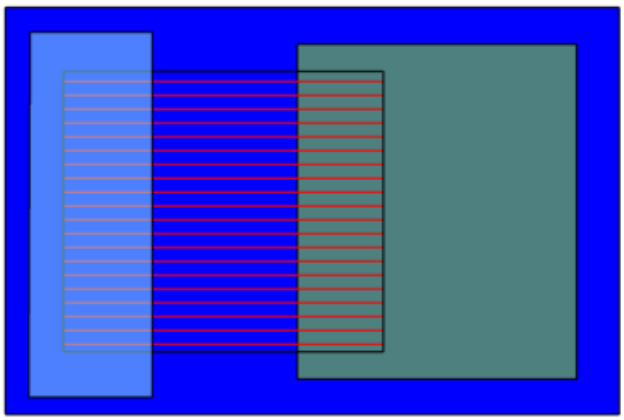
MapLink will attempt to ensure that opaque geometry is drawn before any transparent geometry where possible, but does not provide any guarantees of the order that transparent geometry will be drawn in by default. For situations where the consistently correct display of overlapping transparent objects is important and the transparent geometry cannot be put in a separate render level, the drawing surface allows an application to hint at the required level of correctness through the `TSLOpenGLTransparencyHintEnum` enumeration.

This enumeration lets an application specify the following hints about how it uses transparency on a per-data layer basis, ordered from fastest to slowest:

1. `TSLOpenGLTransparencyHintNever` - MapLink will attempt to draw opaque geometry before transparent geometry where doing so will not cause a significant performance impact. Opaque geometry may be drawn in any order. The rendering order of transparent geometry is not guaranteed.
2. `TSLOpenGLTransparencyHintFlushOpaque` - Any pending opaque geometry not yet rendered will be drawn when a transparent item is encountered. Opaque geometry may be drawn in any order. Transparent items are drawn in order. The performance impact of this setting depends on the number of transparent items in the data layer.
3. `TSLOpenGLTransparencyHintAlways` - No draw reordering occurs - all items are drawn in order from back to front. This setting has a substantial performance impact that increases with the number of items drawn.

In almost all cases using `TSLOpenGLTransparencyHintNever` or `TSLOpenGLTransparencyHintFlushOpaque` is sufficient to give acceptable output - `TSLOpenGLTransparencyHintAlways` should only be used as a last resort due to its performance impact.

Going back to the example at the beginning of this section, the same geometry drawn using the `TSLOpenGLTransparencyHintFlushOpaque` hint will be rendered in the expected order, giving this output:



For systems using OpenGL ES 2.0 or OpenGL 3.1 or earlier, entities using a patterned fill or line style count as transparent for the purposes of draw ordering. On these systems items that would be visible through the pattern will not be shown if they are drawn after the patterned item. When using a system supporting OpenGL 3.2 or later MapLink uses alpha testing for rendering these items and thus they count as opaque geometry for the purposes of draw ordering unless the entity has also been set as transparent via its rendering attributes.

13.9 Anti-aliasing

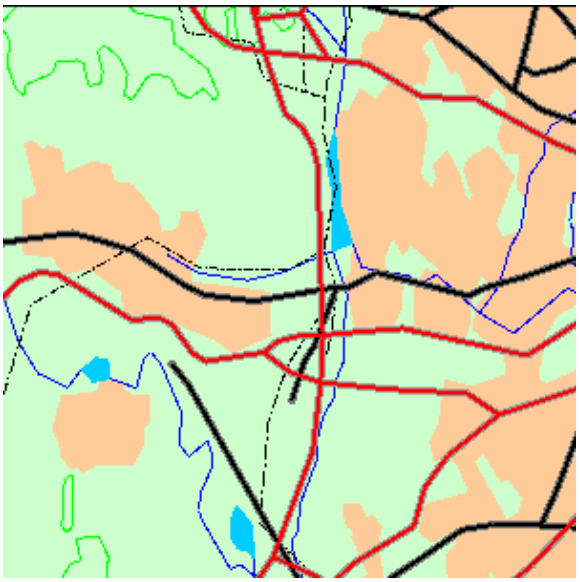
The OpenGL drawing surface can use two different types of anti-aliasing to reduce the visibility of jagged edges on vector features. These are multisampling, which uses features provided by the graphics hardware, and a post-processing implementation that can be used when multisampling is not supported.

13.9.1 Multisampling

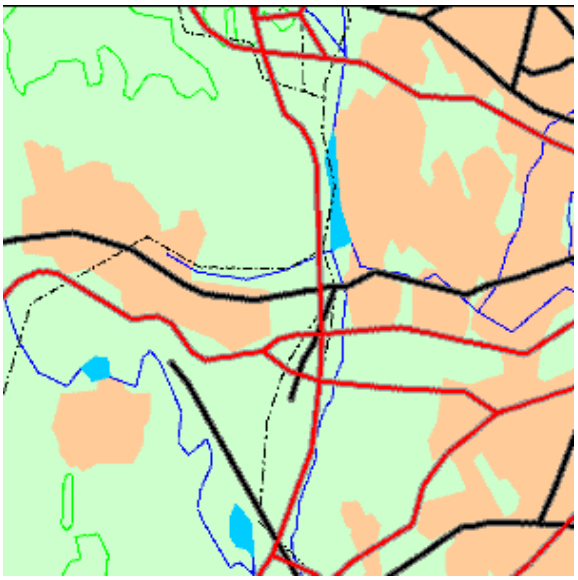
Multisampling can be enabled when creating the drawing surface using one of the constructors that internally create an OpenGL context using the `numMultisampleSamples` option of the `TSLOpenGLSurfaceCreationParameters` class. The drawing surface will attempt to locate a configuration supported by the graphics hardware that supports the requested level of multisampling, but if none exists will use the highest level available on the hardware. Higher levels provide better image quality at the cost of lower performance, but modern hardware can generally run with 4x or higher multisampling without trouble.

The specific levels of multisampling available depend on the graphics hardware in the system.

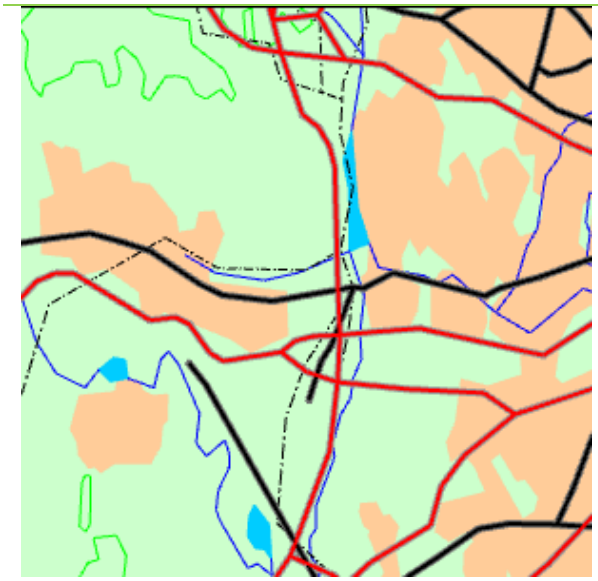
The images shown on the next page illustrate the effect of increasing levels of multisampling on image quality:



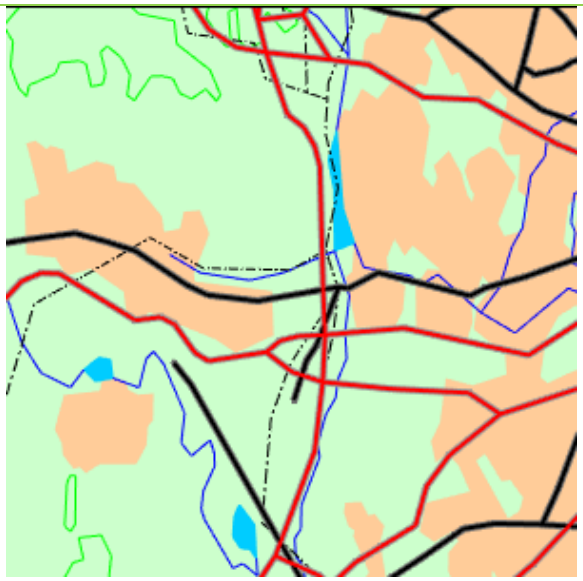
No multisampling



2x multisampling



4x multisampling



8x multisampling

The decision on whether to use multisampling must be made at the time the drawing surface is created - it cannot be enabled or disabled on an existing drawing surface.

On embedded and mobile systems using OpenGL ES 2.0 multisampling will not be applied to any buffered or transparent layers (layers using the `TSLPropertyBuffered` or `TSLPropertyTransparency` data layer properties). On systems that do not support OpenGL 3.2 the ARB_texture_multisample extension is required.

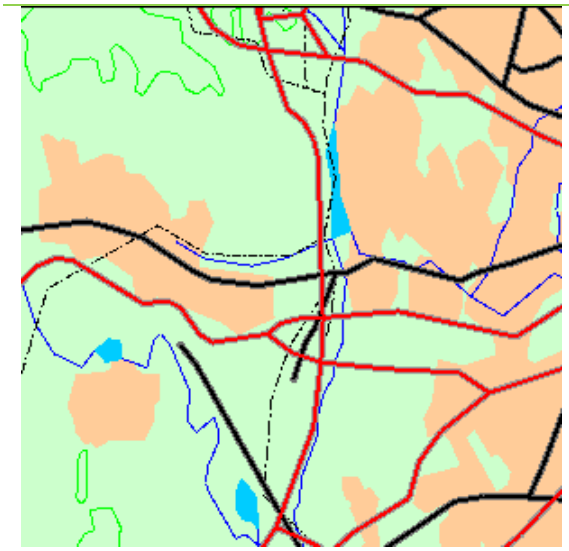
13.9.2 Post-processing Anti-aliasing

When the graphics hardware supports multisampling there is little reason to use post-processing anti-aliasing as multisampling provides better image quality, but if multisampling is not supported then it can be better than no anti-aliasing.

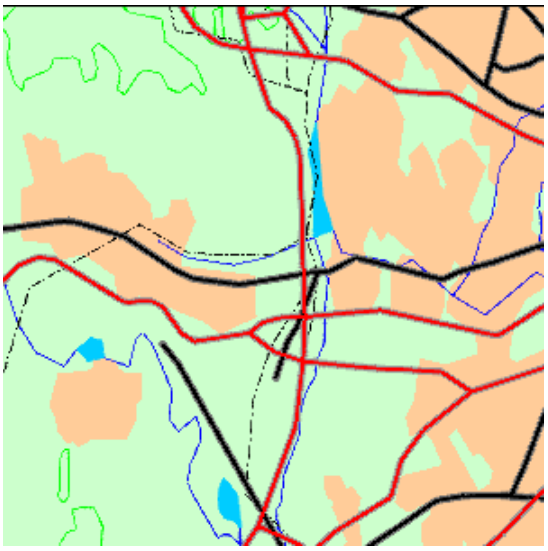
Unlike multisampling, post-processing anti-aliasing can be enabled and disabled at runtime via the `TSLOpenGLSurface::setAntialiasingMode` method as long as the drawing surface was not created with multisampling enabled - multisamping and post-processing anti-aliasing cannot be used at the same time.

Post-processing anti-aliasing has two available settings - FXAAStrong and FXAAWeak. The strong setting will remove most jagged edges but will display noticeable visual artefacts on single pixel thickness features. The weak setting will sometimes leave noticeable jagged edges but does not generate artefacts around single pixel features as the strong setting does.

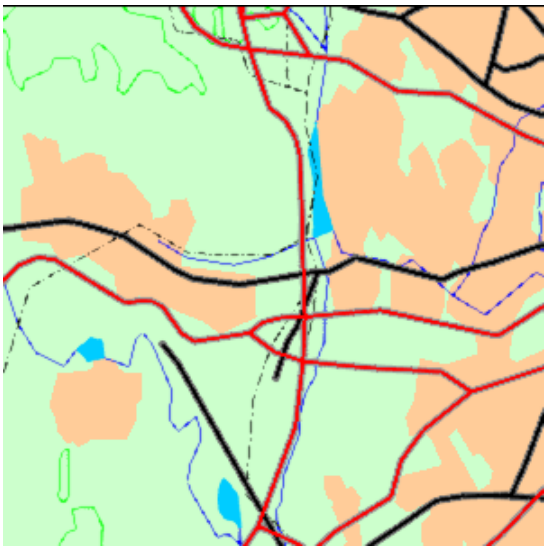
The images below demonstrate the effect of these settings on image quality:



No post-process anti-aliasing



Weak FXAA anti-aliasing



Strong FXAA anti-aliasing

When configuring feature rendering for maps and overlaps that is intended to be used with the post-process anti-aliasing strong setting it is generally better to avoid using any single pixel wide features entirely. Instead configure these features to be two pixels wide and possibly use a slightly lighter colour if the feature may be in areas of high contrast (e.g. black lines against pale backgrounds). The anti-aliasing effect softens the lines so that the extra width is not as pronounced and often provides better quality than using the weak setting with single pixel thick features due to the reduced aliasing.

13.10 Hardware-Supported Raster Formats

Modern graphics hardware supports a number of specialized raster formats (normally referred to as 'compressed textures') that are highly efficient for drawing. These formats are both faster to load and use less memory than other more traditional raster formats such as JPEG and PNG as they do not need to be decompressed before drawing.

In order to allow applications to take advantage of these formats, the `TSLOpenGLDataOptimiser` utility class provides means for an application to convert any

supported raster format into one of the special hardware-supported formats at runtime. These converted rasters can then be loaded as normal into data layers such as the `TSLRasterDataLayer` as with any other raster format.

When displaying very high resolution imagery it is advisable to first generate a raster pyramid from the source raster as described in section 12.8.3 before converting the raster using the `TSLOpenGLDataOptimiser`. Graphics hardware is limited in the maximum resolution of raster that can be displayed (from 2048x2048 on some mobiles and embedded devices; up to 16384x16384 on recent desktop hardware), so creating raster pyramids is necessary to display images that are larger than what is supported by the hardware the application is running on.

Maps created by MapLink Studio that contain rasters can also take advantage of these specialised raster formats when the map is used with the OpenGL drawing surface. This can be done either at the time the map is created by MapLink Studio (refer to the MapLink Studio User Guide for details), or for existing maps through the `TSLOpenGLDataOptimiser`. At runtime MapLink will only attempt to load the image format(s) that are supported by the hardware, so it is not necessary to prepare a version of each map for each image format.

13.11 Integrating with Other OpenGL Applications

It is sometimes desirable to use the OpenGL drawing surface in conjunction with user interface toolkits or other libraries that perform their own OpenGL context creation and management. For this situation each of the window system interface classes provides a constructor that accepts an existing OpenGL context that the drawing surface should use instead of creating its own.

When creating the drawing surface in this way an application usually wants to set the `swapBuffersManually` option from the `TSLOpenGLSurfaceCreationParameters` used to false as buffer swaps are usually managed by the same code that creates the OpenGL context.

To prevent the drawing surface from clearing the colour buffer during a draw, set the clear argument passed to `TSLDrawingSurface::drawDU` or `TSLDrawingSurface::drawUU` to false. This option only inhibits clearing of the colour buffer - the depth buffer and stencil buffer (if present) will still be cleared by the drawing surface.

When changing OpenGL state an application should normally use the drawing surface's `TSLOpenGLStateTracker` object if it provides a function that maps to the state being changed. The drawing surface internally tracks the current OpenGL state in order to remove redundant state changes being sent to the driver which can affect performance; thus it must be kept up to date with the actual OpenGL state in order to function correctly. The initial values of the state tracker are read from OpenGL on drawing surface creation. If the state of any settings that the state tracker stores are modified outside of the state tracker the application should use the `reset` method to force it to re-read the current OpenGL state.

13.11.1 Suggested Framebuffer Configurations

When the OpenGL drawing surface does not create an OpenGL context, it is up to the application to ensure the framebuffer configuration used is suitable for use with the drawing surface. The requirements for the drawing surface are as follows:

- An RGBA framebuffer (`GLX_RENDER_TYPE` is `GLX_RGBA_BIT`, `WGL_PIXEL_TYPE_ARB` is `WGL_TYPE_RGBA_ARB` or `EGL_COLOR_BUFFER_TYPE` is `EGL_RGB_BUFFER`).
- A depth buffer of 16 bits or greater.
- On OpenGL ES 2.0 systems `EGL_RENDERABLE_TYPE` must be `EGL_OPENGL_ES2_BIT`.

Any other settings may be freely chosen by the application.

The `TSLGLXSurface` and `TSLNativeEGLSurface` provide convenience utility methods, named `preferredVisualID` and `preferredConfigID`, that can be used to identify the framebuffer configuration that the drawing surface would prefer to use.

13.12 Off-screen Rendering

There are two main scenarios for wanting to make the OpenGL drawing surface render to an off-screen surface:

1. To take a frame of output from a drawing surface that normally draws to a window and save it for reuse, such as when rendering an overview of a map to a texture.
2. To have a drawing surface that is not attached to a window and always draws off-screen.

Almost no graphics drivers allow for hardware acceleration when rendering to a bitmap or pixmap, therefore the OpenGL drawing surface cannot be attached to one of these for off-screen rendering. Instead, OpenGL itself must be used to accomplish this.

The process for the first type of off-screen rendering is consistent across the various window system interface classes as it only uses standard OpenGL functionality - framebuffer objects, generally referred to as FBOs.

The second type of off-screen rendering affects how the OpenGL context must be created, and thus the specifics differ between Windows, X11 and EGL based systems.

13.12.1 Redirecting Drawing Surface Output to a Framebuffer Object

The OpenGL drawing surface respects any framebuffer object (FBO) bound at the point that it starts drawing and will redirect its output to the buffers bound to this FBO. Therefore making the drawing surface render to a texture or renderbuffer is as simple as binding the desired FBO before calling `drawDU` or `drawUU` on the drawing surface. The code to create this FBO would look as follows:

```
// m_surface is a TSLOpenGLSurface that is attached to a window as
// normal for on-screen rendering.

// Creates the framebuffer object and attachments to use for offscreen
// rendering
bool createFBO()
{
  TSLDeviceUnits surfaceX1 = 0, surfaceY1 = 0,
                 surfaceX2 = 0, surfaceY2 = 0;
  m_surface->getDUExtent( &surfaceX1, &surfaceY1,
                          &surfaceX2, &surfaceY2 );

  glGenFramebuffers( 1, &m_fbo );
  glGenTextures( 1, &m_texture );
  glGenRenderbuffers( 1, &m_depthBuffer );

  // Initialise the colour texture and depth render buffer to
  // match the drawing surface's size
  m_surface->stateTracker()->bindTexture( GL_TEXTURE0, GL_TEXTURE_2D,
                                          m_texture );
  glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST );
  glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST );
  glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE );
  glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE );
  glTexImage2D( GL_TEXTURE_2D, 0, GL_RGBA8, surfaceX2-surfaceX1,
                surfaceY2-surfaceY1, 0, GL_RGBA, GL_UNSIGNED_BYTE,
                NULL );

  glBindRenderbuffer( GL_RENDERBUFFER, m_depthBuffer );
  glRenderbufferStorage( GL_RENDERBUFFER, GL_DEPTH_COMPONENT24,
                         surfaceX2-surfaceX1, surfaceY2-surfaceY1 );

  // Make the fbo the active render target
  m_surface->stateTracker()->bindFramebuffer( GL_FRAMEBUFFER, m_fbo );
  glFramebufferTexture2D( GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
                          GL_TEXTURE_2D, m_texture, 0 );
  glFramebufferRenderbuffer( GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
                             GL_RENDERBUFFER, m_depthBuffer );

  // Make sure we created the fbo correctly
  GLenum status = glCheckFramebufferStatus( GL_FRAMEBUFFER );
  switch( status )
  {
    case GL_FRAMEBUFFER_COMPLETE:
      m_surface->stateTracker()->bindFramebuffer( GL_FRAMEBUFFER, 0 );
      // Everything worked - we are done
      return true;

    default:
       // An error occurred - the created objects should be deleted here
       ...
       return false;
  }
}
```

To make the drawing surface draw to the texture the applications draw function might look as follows:

```
// m_surface is a TSLOpenGLSurface that is attached to a window as
// normal for on-screen rendering.
//
// m_fbo is the framebuffer object already created

void draw( int width, int height, bool drawOffscreen )
{
  if( drawOffscreen )
  {
    m_surface->stateTracker()->bindFramebuffer( GL_FRAMEBUFFER, m_fbo );
    // Drawing will go to the buffers attached to the FBO - in this
    // case the texture created earlier.
    m_surface->drawDU( 0, 0, width, height, true );
  }
  else
  {
    m_surface->stateTracker()->bindFramebuffer( GL_FRAMEBUFFER, 0 );
    // Drawing will go to the window the surface is attached to
    m_surface->drawDU( 0, 0, width, height, true );
  }
}
```

After the draw completes the texture will then contain the output from the drawing surface.

13.12.2 Windowless Drawing Through GLX with the TSLGLXSurface

GLX allows an OpenGL context to be created from a `GLXPbuffer` instead of a `Window` XID. These are created through `glXCreatePbuffer` which returns the `GLXDrawable` needed to create the OpenGL context, which the drawing surface can then be attached to. When using `GLXPbuffer`s the application must create the OpenGL context itself. An example of this is below:

```
int configAttribs[] = { GLX_DOUBLEBUFFER, False,
                        GLX_DEPTH_SIZE, 16,
                        GLX_DRAWABLE_TYPE, GLX_PBUFFER_BIT,
                        GLX_RENDER_TYPE, GLX_RGBA_BIT,
                        GLX_CONFIG_CAVEAT, GLX_NONE,
                        None };
int numConfigMatches = 0;
GLXFBConfig *configs = glXChooseFBConfig( m_display, m_screenNum,
                                          configAttribs,
                                          &numConfigMatches );
if( !configs || numConfigMatches == 0 )
{
  // Error - no valid framebuffer configurations
  ...
}

int bufferAttribs[] = { GLX_PBUFFER_WIDTH, width,
                        GLX_PBUFFER_HEIGHT, height,
                        None };
GLXPbuffer glxDrawable = glXCreatePbuffer( m_display, configs[0],
                                           bufferAttribs );

// This code assumes support for GLX_ARB_create_context
int contextAttribs[] = { GLX_CONTEXT_MAJOR_VERSION_ARB, 3,
                         GLX_CONTEXT_MINOR_VERSION_ARB, 2,
                         GLX_CONTEXT_PROFILE_MASK_ARB,
                         GLX_CONTEXT_CORE_PROFILE_BIT_ARB
                         GLX_RENDER_TYPE, GLX_RGBA_TYPE,
                         None };
GLXContext context = glXCreateContextAttribs( m_display, configs[0], None,
                                              True, contextAttribs);

if( !glXMakeContextCurrent( m_display, glxDrawable, glxDrawable,
                            context ) )
{
  // Error - cannot activate context
  ...
}

// Attach the drawing surface to the OpenGL context
TSLOpenGLSurfaceCreationParameters creationOptions;
creationOptions.useVSync( false );
creationOptions.swapBuffersManually( true );
TSLGLXSurface *surface = new TSLGLXSurface( m_display, m_screen,
                                            glxDrawable, context,
                                            creationOptions );

XFree( configs );
```

13.12.3 Windowless Drawing Through EGL with the TSLEGLSurface

Windowless rendering with EGL is very similar to GLX - an OpenGL context can be created from an `EGLSurface` tied to a Pbuffer instead of an `EGLNativeWindowType`. When using EGL Pbuffers the application must create the OpenGL context itself. An example of this is below:

```
EGLint configAttribs[] = { EGL_RENDERABLE_TYPE, EGL_OPENGL_ES2_BIT,
                           EGL_COLOR_BUFFER_TYPE, EGL_RGB_BUFFER,,
                           EGL_SURFACE_TYPE, EGL_PBUFFER_BIT,
                           EGL_DEPTH_SIZE, 16,
                           EGL_SAMPLES, 0,
                           EGL_NONE };
EGLint numConfigMatches = 0;
EGLConfig config;
if( !eglChooseConfig( m_display, configAttribs, &config, 1,
                      &numConfigMatches ) || numConfigMatches == 0 )
{
 // Error - no valid framebuffer configurations
  ...
}

EGLint bufferAttribs[] = { EGL_WIDTH, width,
                           EGL_HEIGHT, height,
                           EGL_NONE };
EGLsurface eglSurface = eglCreatePbufferSurface( m_display, config
                                                  bufferAttribs );

EGLint contextAttribs[] = { EGL_CONTEXT_CLIENT_VERSION, 2,
                            EGL_NONE };
EGLContext context = eglCreateContext ( m_display, config, EGL_NO_CONTEXT,
                                        contextAttribs );

if( !eglMakeCurrent( m_display, eglSurface, eglSurface, context ) )
{
  // Error - cannot activate context
  ...
}

// Attach the drawing surface to the OpenGL context
TSLEGLSurface *surface = new TSLEGLSurface();
surface->attach();
```

13.12.4 Windowless Drawing on Windows with the TSLWGLSurface

Windowless drawing on Windows is not possible - while WGL has the same pbuffer concept as GLX and WGL through the `WGL_ARB_pbuffer` OpenGL extension, because it is an OpenGL extension the functions necessary to use it require an active OpenGL context - thus giving a circular dependency of requiring an OpenGL context in order to create an OpenGL context.

The standard method of dealing with this situation is to create an invisible window to create the OpenGL context from, and use the FBO method described in section 13.12.1. Applications should not render to the invisible window as this is undefined behaviour in OpenGL and so may not work on all graphics hardware.

13.13 Threading

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.

The thread the drawing surface's context is bound to can be changed by calling the `makeContextCurrent` function available on each of the window system interface classes from the new thread, however this is an expensive operation so applications should avoid frequently migrating a drawing surface between threads.
Drawing data layers and entities in the OpenGL drawing surface creates GPU resources than 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 or using the `releaseResources` method on the `TSLDataLayer` or `TSLEntity`.

A multithreaded application should generally assign one rendering thread to each OpenGL context (and thus drawing surface) used, with other application logic occurring other threads. Guidelines on using MapLink in a threaded environment can be found in section 28.

13.14 Performance Tips

There are several important considerations when writing an application using the OpenGL drawing surface in order to achieve maximum performance. This section contains suggestions on how to approach various tasks in the most performant way with the OpenGL drawing surface and lists some common pitfalls to avoid.

13.14.1 General Tips

The first time an item is drawn the OpenGL drawing surface will perform a set of processing tasks necessary in order to create the necessary GPU resources to draw the item. The results of this processing are associated with the item so that they can be reused in subsequent draws. For complex items this calculation can take a noticeable amount of time and so an application should try and avoid situations that cause this calculation to be redone unnecessarily. The situations that can trigger this recalculation are as follows:

Removing a `TSLDataLayer` from the drawing surface will delete any associated GPU resources. Re-adding the data layer to the surface will trigger the processing task for the data layer on the next draw. Applications should avoid removing and re-adding data layers to the surface as a method of controlling layer visibility - the `TSLPropertyVisible` data layer property should be used for this task.

Calling `notifyChanged` on a `TSLDataLayer` indicates to the drawing surface that the any existing resources need to be recreated due to the underlying data of the data layer changing in a way it cannot detect. In most cases the drawing surface can identify when the data layer has been modified in a way that requires recalculation (such as loading a

new map into a `TSLMapDataLayer` or modifying the contents of a `TSLEntitySet`), so applications should only call `notifyChanged` in situations where this will not occur automatically.

Modifying a `TSLEntity` by changing its coordinates, or in the case of a `TSLEntitySet` adding/removing entities from the set invalidates the GPU resources for all entities in the controlling data layer or entity set. The OpenGL drawing surface offers per-data layer controls for applications to provide hints on how GPU resources should be created for `TSLEntities` in order to minimise the amount of recalculations that occur - see section 13.14.2 for details.

Each transparent data layer in a draw uses an additional amount of graphics memory equal to that used by the default framebuffer of the drawing surface. Applications should avoid large numbers of transparent layers on graphics hardware with low amounts of memory.

Applications should use the drawing surface's `TSLOpenGLStateTracker` to modify the OpenGL state where possible instead of using its `reset` method to force it to re-read the OpenGL state. Re-reading the OpenGL state will normally cause the thread containing the OpenGL context to block until all pending OpenGL commands are complete which can cause stalls during application rendering.

Data layers shared between multiple OpenGL drawing surfaces do not share GPU resources even if their OpenGL contexts share resource lists. Therefore, a data layer in two OpenGL drawing surfaces requires twice as much graphics memory as the same data layer in one drawing surface.

Increasing the cache size of a `TSLDataLayer` will increase the amount of GPU memory used accordingly. Applications should try to avoid setting large cache sizes on systems that have low amounts of GPU memory.

Other documentation may refer to performance differences between cosmetic and geometric pens when drawing lines. The OpenGL drawing surface has no distinction between these types and therefore there is no performance difference between then.

13.14.2 Vector Geometry in Data Layers

By default, data layers containing vector geometry (i.e. `TSLEntities`) have one combined GPU resource created for all entities within a single tile of that layer, or one resource for that layer for layers that are not tiled.

When using layers providing user-editable `TSLEntities`, such as the `TSLStandardDataLayer` this can be problematic since changing any single `TSLEntity` requires reprocessing the entire tile or entity set. For this situation the OpenGL drawing surface allows an application to provide a hint about how it should structure GPU resources to avoid unnecessary recalculation through the `TSLOpenGLStorageStrategyEnum` enum and `setLayerStorageStrategy` method on `TSLOpenGLDrawingSurface`. This allows an application to structure its data into a series of self-contained resource groups that are modified together.

The diagrams below show a simple entity hierarchy using each of the storage strategies. The background colour of each item denotes the resource group it belongs to, with items having the same background colour being in the same group.

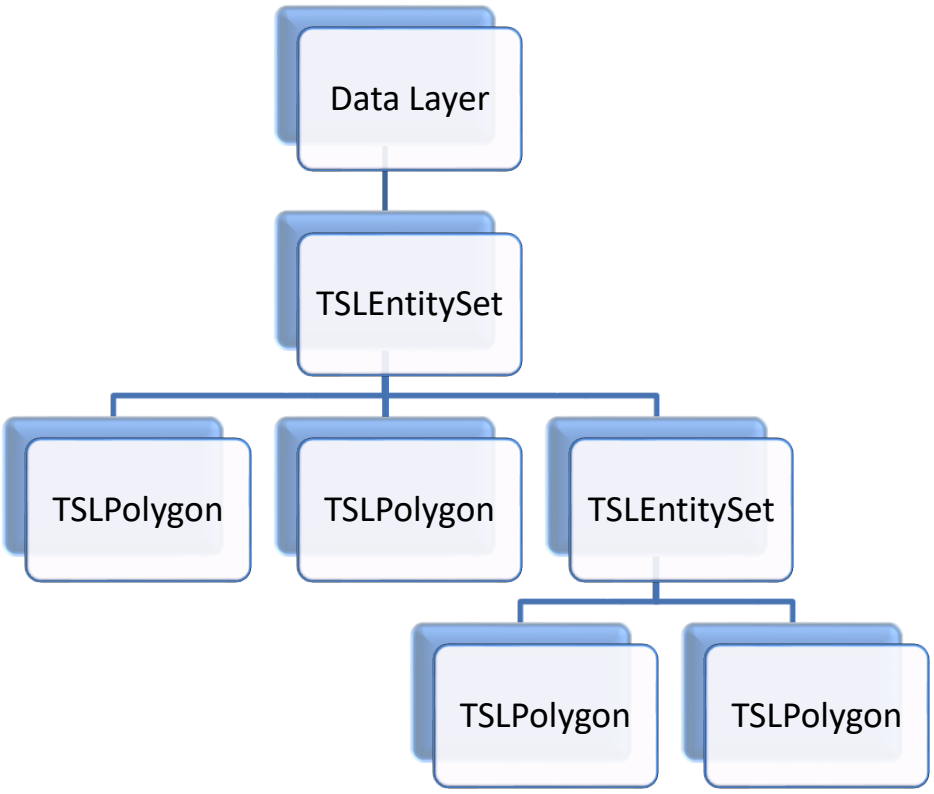


Figure 20 - Per Tile Storage Strategy

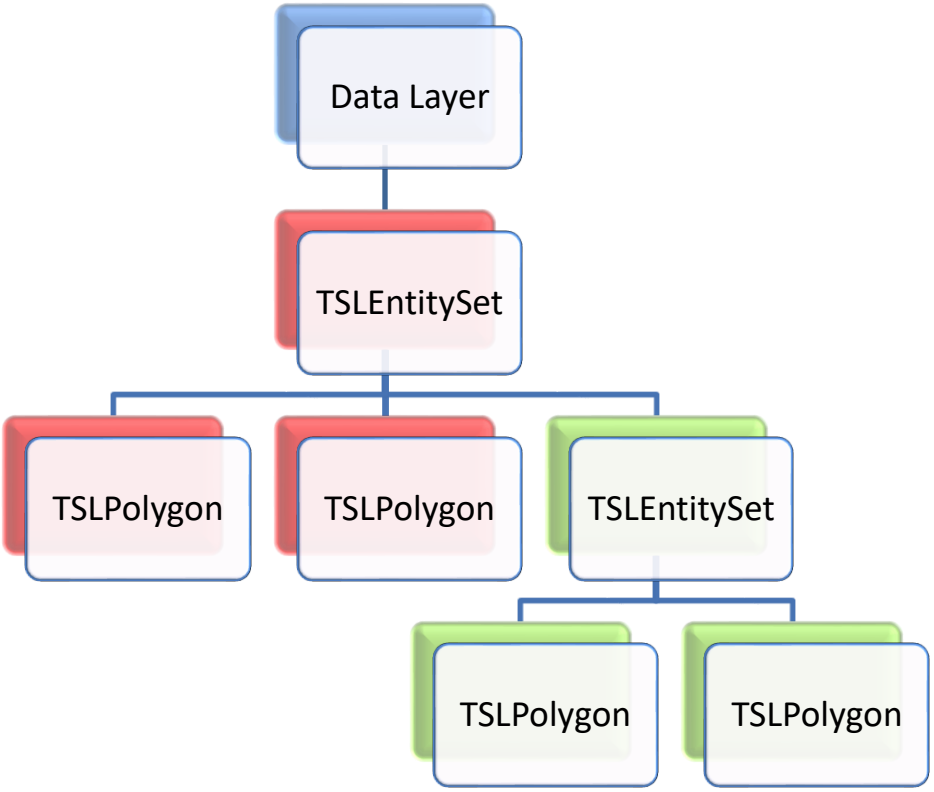


Figure 21 - Per Entity Set Storage Strategy

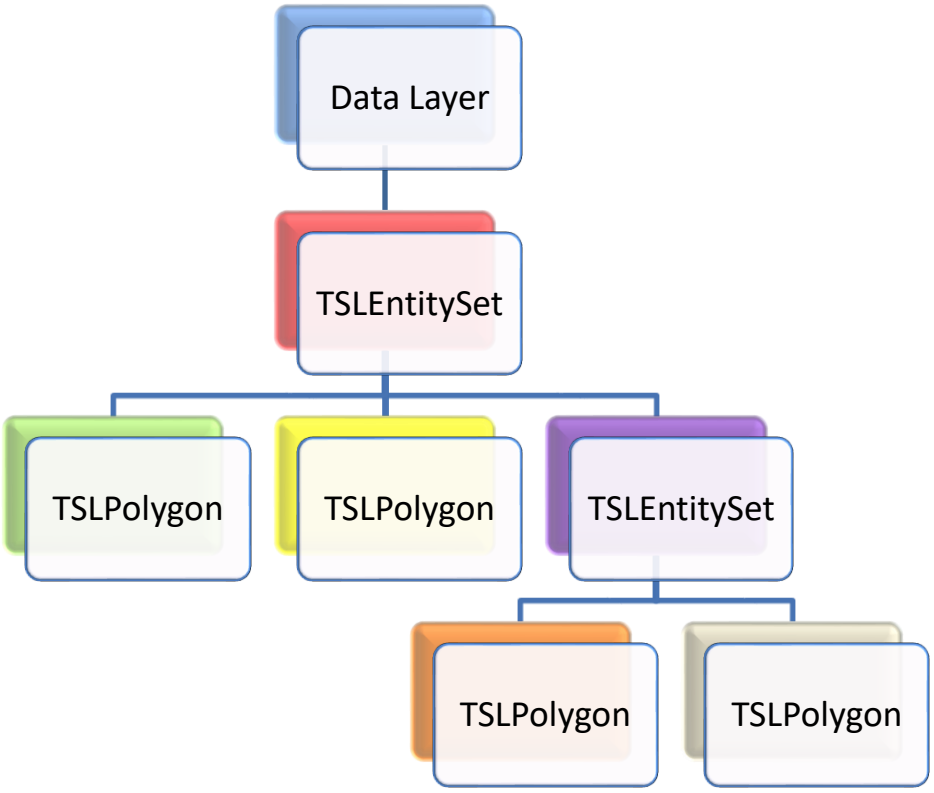


Figure 22 - Per Entity Storage Strategy

If an application had two types of vector geometry in a `TSLStandardDataLayer`, one that was rarely modified and one that was frequently modified, these could be stored in two different `TSLEntitySet`s on the `TSLStandardDataLayer`, with the layer's storage strategy set to `TSLOpenGLPerEntitySetStrategy`. This way, modifying the contents of one entity set would not invalidate the GPU resources of the other entity set leading to improved performance.

Applications should avoid creating many small resource groups in this fashion as each additional group requires additional OpenGL state changes to draw and reduces the ability for the drawing surface to reorder drawing for best performance. For the same reason an application should avoid using the `TSLOpenGLPerEntityStrategy` unless absolutely necessary.
Using the `move`, `rotate`, `rotation`, `scale`, `scaleXY` and `translate` methods on `TSLEntity`, or modifying the rendering attributes of the `TSLEntity` does *not* trigger the invalidation of GPU resources for the resource group.

13.14.3 Using the TSLRenderingInterface

Applications should avoid using `drawPolygon` and `drawPolyline` on the `TSLRenderingInterface` in conjunction with the OpenGL drawing surface. These methods require the drawing surface to create and delete the necessary resources to draw the geometry every time the method is called, which leads to increasingly poor performance the more these methods are called in a frame.

Instead, an application should set the layer storage strategy on the drawing surface for the data layer to `TSLOpenGLPerEntitySetStrategy` and create a persistent `TSLEntitySet` that contains `TSLPolygon`s and `TSLPolyline`s for each of the items that need to be drawn. The application should draw these items by passing the containing `TSLEntitySet` to the `drawEntity` function on the `TSLRenderingInterface`. This allows the drawing surface to avoid having to perform the same set of calculations every frame for this geometry.

For the same reasons an application should not recreate any `TSLEntities` that will be used for drawing each frame - these should be persisted between frames where possible.

Any entities used with the `TSLRenderingInterface` in this fashion must have the `releaseResources` method called from the application's drawing thread before the drawing surface is destroyed or detached.

13.14.4 Dynamic Renderers

Dynamic renderers used with the OpenGL drawing surface should avoid calling `drawPolygon` and `drawPolyline` on the `TSLRenderingInterface` as described in section 13.14.3. When using a dynamic renderer to draw an entity with custom rendering attributes the renderer should always return `TSLDynamicRendererActionUseCurrentRendering` from the `render` method and allow MapLink to draw the entity. Dynamic renderers implemented this way will perform substantially faster when used with the OpenGL drawing surface.

13.14.5 Raster Data in Data Layers

Raster drawing is normally fast in all circumstances and there is little an application needs to do to ensure good performance. Some general information is presented below:

Palletized rasters are slower to draw than RGB rasters. An application should not convert RGB images to palletized images on the expectation of increased drawing performance.

Rasters must be decompressed in order to be drawn. This can consume a substantial amount of GPU memory if many rasters are used at the same time. The application can use the cache size setting of the data layer to control the amount of memory used by the data layer.

- Excessive minification of rasters (drawing a raster at a significantly lower resolution than the image itself) can be slow on some hardware, especially when the raster approaches the maximum supported resolution of the hardware. Applications can generate raster pyramids in these cases using `TSLRasterUtilityFunctions::rasterToPyramid` for runtime loaded rasters, or enabling raster pyramiding in MapLink Studio when generating raster maps.
- Mobile and embedded devices usually have very limited fill rates, meaning each pixel on the screen cannot be updated may times before drawing performance slows down. On these devices an application should avoid layering multiple rasters on top of each other when the underlying rasters are not visible.

For best performance applications should make use of the compressed texture format support described in section 13.10 when possible, particularly on mobile or embedded platforms when GPU memory is limited.

13.14.6 Map Creation Guidelines

The OpenGL drawing surface, especially on mobile or embedded devices, is more susceptible to performance problems from poorly constructed maps. In particular, users should read the 'Optimising Maps for Performance' section from the MapLink Studio user guide before creating their map.

Vector maps intended for use with the `TSLStaticMapDataLayer` (see section 13.6) should be created with the vector processing option enabled as described in the 'Platform Specific Tips – Embedded, Mobile and OpenGL' section of the MapLink Studio user guide. Doing so removes the need for the data layer to perform the same processing when loading new data for display which substantially improves application responsiveness when zooming and panning the map.

Raster maps should generally always be created with one or more of the compressed raster options enabled as described in the 'Platform Specific Tips – Embedded, Mobile and OpenGL' section of the MapLink Studio user guide. The specific formats to use depends on the capabilities of the target hardware that will be used to display the map.

When assessing the complexity of a vector map it can be useful to put the drawing surface into wireframe mode. This can provide a quick visual method of determining if a

particular map layer is unnecessarily detailed for the scale it is displayed at. This can be done by inserting the following line of code before calling `drawDU`:

```
glPolygonMode( GL_FRONT_AND_BACK, GL_LINE );
```

As a general rule, areas that appear as mostly solid colour in wireframe mode when viewing a detail layer at its highest zoom level are unnecessarily detailed and could be simplified without affecting the appearance of the detail layer.

When creating maps for mobile or embedded devices it can often be more efficient to use the background colour setting in MapLink Studio to colour large areas of ocean or land instead of using a background rectangle if the actual features are not required at runtime. This can significantly improve performance on some devices that have limited fill rates.

13.15 Behavioural Differences to Other Drawing Surfaces

In most cases the MapLink 2D drawing surfaces can be treated interchangeably, once created most interactions occur through the `TSLDrawingSurface` and `TSLDrawingSurfaceBase` classes which provide consistent behaviour regardless of the specific surface type.

While this is generally true for the OpenGL surface as well, the underlying technology it uses operates substantially differently to the other MapLink drawing surfaces and so there are inevitably some areas where it will behave differently to the `TSLNTSurface` or `TSLMotifSurface`. This section is intended to provide a summary of these differences to developers already familiar with the behaviour of the `TSLNTSurface` or `TSLMotifSurface` - all the information below is also listed in the API documentation for the appropriate classes/functions.

The origin for `TSLDrawingSurface::wndResize` is bottom left rather than top left. Note that all other functions that take `TSLDeviceUnits` still operate with a top-left origin (e.g. `TSLDrawingSurface::drawDU`). When sizing the drawing surface to cover the entire window it is attached to this difference has no practical effect - the arguments passed are the same. Using `wndResize` to restrict the drawing surface to part of a window is discouraged.

The `updateExtentOnly` flag on `TSLDrawingSurface::drawDU` and `TSLDrawingSurface::drawUU` causes any drawing, including the clear operation if `clear` is set to true, to be hard clipped to the given region in the OpenGL surface. With the `TSLNTSurface` and `TSLMotifSurface` drawing can still occur outside the designated region.

Drawing can only occur in the thread that the OpenGL context is bound to - by default this is the thread the drawing surface is created in. See section 13.13 for more details.

The drawing surface cannot be attached to a bitmap or pixmap for off-screen rendering unless this is explicitly supported by the graphics driver - most drivers do not support this. See section 13.12 for details on how to perform off-screen rendering with the OpenGL drawing surface.

The drawing surface frame that can be activated via `TSLDrawingSurface::setFrame` is not supported.

ROP fillstyles (style IDs 600-614) are not supported.

The `TSLOptionDoubleBuffered` drawing surface option has no effect on the drawing surface. Double buffering is determined during OpenGL context creation and cannot be changed afterwards.

The `TSLOptionAntiAliasMonoRasters` drawing surface option has no effect on the drawing surface. Anti-aliasing is determined during OpenGL context creation and cannot be changed afterwards. Any anti-aliasing applies to all rendering performed via the drawing surface. See section 13.9 for more details.

MapLink OS-specific types are undefined for the OpenGL drawing surface, e.g. `TSLDeviceContext`, `TSLWindowHandle`. Functions which accept these types cannot be used in conjunction with the OpenGL drawing surface. Equivalent functions are provided on the window system interface classes where appropriate.

The OpenGL drawing surface cannot be cloned. The clone method will always return NULL.

Hershey vector fonts as described in section 12.6.8 are not supported as they would be substantially slower than the normal font rendering.

Text is always rendered with anti-aliasing enabled as there is no performance overhead in doing so.

OpenGL 3.2 is required for mitre and bevel line joins. When this is not available any lines using these join types will be drawn with rounded joins instead.

13.16 Migrating from Other Drawing Surfaces

Updating an existing application using one of the other MapLink drawing surfaces to use the OpenGL drawing surface is quite simple. This involves replacing the drawing surface class used by the application with the applicable window system interface class for the target platform (i.e. `TSLNTSurface` is replaced with `TSLWGLSurface`, and `TSLMotifSurface` is replaced with `TSLGLXSurface`). As the OpenGL drawing surface implements the same interface as the other MapLink drawing surfaces this should not affect the majority of the application's code if the drawing surface is referenced as a `TSLDrawingSurface` in the sections of the application that do not deal with the windowing system.

On X11 systems the visual used to create the window that the drawing surface is attached to determines the properties of the framebuffer used by the drawing surface (such as the presence of double buffering), therefore it is important to use a visual suitable for the drawing surface. The `TSLGLXSurface` provides a helper method called `preferredVisualID` to assist the application is selecting a suitable visual.

In practice, this means that an application's original drawing surface creation that looks like this:

```
// On Windows
TSLNTSurface *surface = new TSLNTSurface(m_hWnd, false);
surface->setOption(TSLOptionDoubleBuffered, true);
surface->wndResize(0, 0, width, height, false);

... // Non-drawing surface application initialisation
```

```
// On X11
// Find the visual to use when creating the window
Visual *visual = applicationVisualChooser();

// Creates the X window using the chosen visual
Window window = createWindow(visual);

TSLMotifSurface *surface = new TSLMotifSurface(display, screen,
                                               colourmap, window, 0,
                                               visual);
surface->setOption(TSLOptionDoubleBuffered, true);
surface->wndResize(0, 0, width, height, false);

... // Non-drawing surface application initialisation
```

Would be replaced with the following:

```
// On Windows
TSLOpenGLSurfaceCreationParameters creationOptions;
TSLWGLSurface *surface = new TSLWGLSurface(m_hWnd, false,
                                           creationOptions);
m_drawingSurface->wndResize(0, 0, width, height);

... // Non-drawing surface application initialisation
```

```
// On X11
// Find the visual to use when creating the window
TSLOpenGLSurfaceCreationParameters creationOptions;
int visualID = TSLGLXSurface::preferredVisualID(display, screen,
                                                creationOptions);

XVisualInfo visualTemplate;
visualTemplate.screen = screenNum;
visualTemplate.visualid = visualID;
int numVisualMatches = 0;
XVisualInfo *visualData = XGetVisualInfo(display, VisualIDMask |
                                         VisualScreenMask,
                                         &visualTemplate,
                                         &numVisualMatches);

// Creates the X window using the chosen visual
Window window = createWindow(visualData->visual);

TSLGLXSurface *surface = new TSLGLXSurface(display, screen, window,
                                           visualData->visual,
                                           creationOptions);
m_drawingSurface->wndResize(0, 0, width, height);

XFree(visualData);

... // Non-drawing surface application initialisation
```

13.16.1 Interaction Modes

Previously MapLink provided two interaction mode manager classes, one for use on Windows that uses GDI to render feedback information and one for use on X11 systems that uses Xlib to render feedback information.

Using these interaction mode managers in conjunction with the OpenGL drawing surface can be problematic as this requires the graphics driver of the system to allow mixed OpenGL and GDI/Xlib rendering to the same window and not all implementations provide this capability.

To solve this issue a new interaction mode manager, `TSLInteractionModeManagerGeneric`, and its associated display class are provided that use the underlying drawing technology of the drawing surface it is created with. This manager can be used with any of the MapLink drawing surfaces.

13.16.2 Applications Containing Custom GDI or Xlib Rendering

Applications that contain custom drawing code, e.g. in a custom data layer or Dynamic Data Objects, using GDI on Windows or Xlib on X11 systems will require additional changes in order to use the OpenGL drawing surface. Even on systems where mixing OpenGL and GDI/Xlib rendering is supported, any drawing that the application performs in this fashion will go directly to the framebuffer's front buffer, not the back buffer that the drawing surface is drawing to. This means that once a buffer swap occurs at the end of a draw this rendering will be overwritten by the contents of the back buffer and will not be seen by the user.

In order to assist applications with migrating to using the OpenGL drawing surface MapLink provides a concept called *non-native rendering* (sometimes also referred to as *non-native drawing*). Effectively this allows an application to use a drawing technology, generally GDI or Xlib, in custom layers different to the one used by the drawing surface without the problems mentioned above.

To use non-native rendering, an application should set the `TSLPropertyNonNativeDrawing` data layer property for each layer that does not draw using OpenGL. It should be noted that a single layer should not try to mix both OpenGL and non-native drawing as the results will not be as expected - for this reason the `draw` methods on the `TSLRenderingInterface` will be disabled during the drawing of a non-native data layer.
Applications should note that the drawing surface provided non-native drawing target will always be cleared each time it is drawn, regardless of the value of the `clear` flag passed to the drawing surface's `drawDU` method. This is because in some implementations the target is reused between multiple non-native layers and thus not clearing the contents would lead to unexpected output.

Users of this functionality should be aware that there is a significant performance overhead associated with merging the output of different drawing technologies. This capability is provided to allow for reuse of existing code and should not be used as a basis for development of significant amounts of new non-native drawing code. An application that was performance limited by the speed of non-native custom drawing with one of the other MapLink drawing surfaces will usually run slightly slower using the OpenGL drawing surface due to the extra synchronisation necessary.

It is recommended for best performance that the application's custom rendering is rewritten in OpenGL where possible.

13.16.3 Non-Native Layers on Windows

When drawing a non-native layer on Windows, the drawing surface creates a memory bitmap for the application to draw to. For the duration of the layer's `draw` method the `handleToDrawable` method of the `TSLRenderingInterface` will return an HDC with this bitmap already selected into, which the application can draw directly into using GDI or similar. This is almost identical to how the `TSLNTSurface` behaves, with the exception that the HDC returned is never the same as the one the drawing surface was attached to.

By default the drawing surface assumes that the application will use GDI to draw to the bitmap. As GDI does not support alpha as a colour component, the drawing surface assumes that any pixels drawn to as part of the layer's drawing will have the alpha component set to 0. When merging the non-native drawing onto the framebuffer, pixels whose alpha value are 255 will not be displayed, while pixels whose alpha value is 0 will be shown.

If the application uses drawing methods that properly support alpha, such as those from GDI+, the default setting will result incorrect rendering. In this case the `enableNonNativeGDIAlphaCorrection` method on `TSLWGLSurface` should be used to disable the automatic inversion of the alpha channel of the bitmap.

13.16.4 Non-Native Layers on X11

The use of non-native layers on X11 systems requires support for the `GLX_EXT_texture_from_pixmap` OpenGL extension and the Xrender X11 extension.

When drawing a non-native layer on X11 systems, the drawing surface creates a pixmap for each non-native layer in the drawing surface. For the duration of the layer's `draw` method the `handleToDrawable` and `handleToX11DrawingParameters` methods of the `TSLRenderingInterface` will return the drawable and visual that the application should draw into. The drawable will always be a 32bit pixmap. It should be noted that the visual returned may be different from the one used when creating the drawing surface, especially if multisampling is in use.

Application drawing to the pixmap should correctly set the alpha channel in order for the results to be visible. If using the base Xlib drawing functions that do not directly allow the specification of the alpha component, this can be done through direct manipulation of the pixel value on True Colour visuals like so:

```
XVisualInfo visualTemplate;
visualTemplate.visualid = XVisualIDFromVisual( visual );
int numMatches = 0;
XVisualInfo *visualInfo = XGetVisualInfo( display, VisualIDMask,
                                          &visualTemplate, &numMatches );

unsigned long alphaBytes = std::numeric_limits<unsigned long>::max();
alphaBytes ^= (visualInfo->red_mask | visualInfo->green_mask |
               visualInfo->blue_mask);
XFree( visualInfo );

// Determine normal pixel value as normal through either XAllocColor
// or calculation based on the visual's colour masks
XColor colour = ...

unsigned long alphaColour = alphaBytes | colour.pixel;
```

14 DIRECT IMPORT SDK

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/vector from a single file. The layer provides the ability to reproject 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.

14.1 Library Usage and Configuration

As of version 11.1, MapLink is no longer supplied with Debug or 32-bit libraries. Therefore, your application's build should link against the Release Mode libraries in all configurations.

MapLinkDirectImport64.lib
Release mode, DLL version.
Uses Multithreaded DLL C++ run-time library.

Must also link the MapLink CoreSDK library MapLink.lib/MapLink64.lib

Requires TTLDLL preprocessor directive.

Refer to the document "MapLink Pro X.Y: Deployment of End User Applications" for a list of run-time dependencies when redistributing. Where X.Y is the version of MapLink you are deploying.

14.2 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`.

Current formats supported are listed in Appendix B.

14.3 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.

In order to load a dataset 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. In order 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 in order to maintain performance.

14.4 Data Processing and Display

When a data set is loaded into the layer it will be split into several 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 in order to pre-process the data into the on-disk cache.

14.5 Callbacks

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

In order 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.

14.6 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 the source data. This should be performed as an offline process, in order to produce a feature configuration for the data or product. Alternatively, the feature configuration may be exported from the MapLink Studio feature book.

14.7 Raster Specific Settings

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

14.8 Caching

14.8.1 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 IO when panning the map display.

14.8.2 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.

14.8.3 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.

14.9 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.

14.9.1 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.

14.9.2 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
```

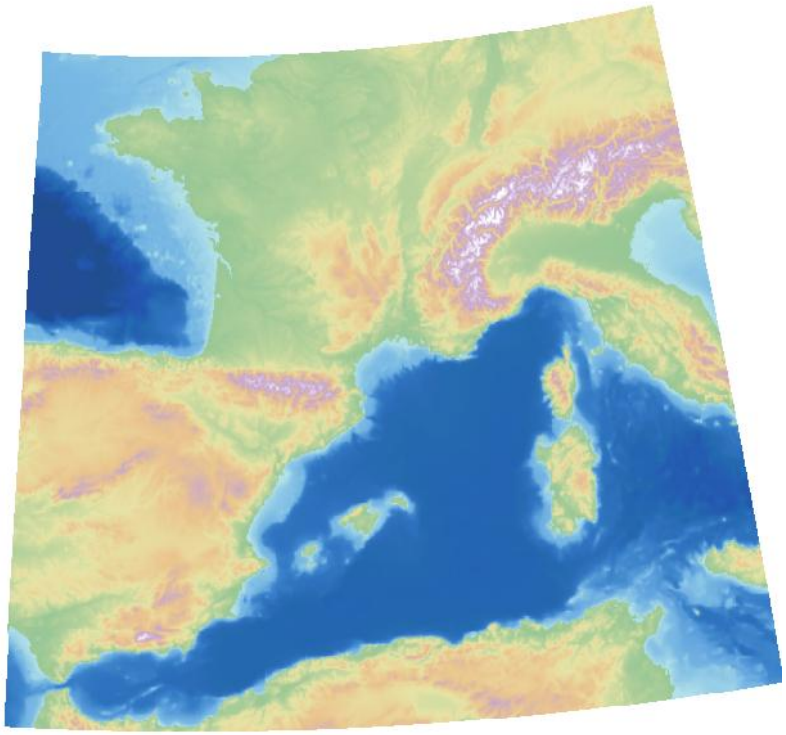
Combine those into a single file that can be loaded into the Direct Import Data Layer:

```
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:



14.10 Direct Import Drivers

Direct Import 'drivers' provide support for each data format via a plugin architecture. When created the `TSLDirectImportDataLayer` will load all available drivers.

These libraries must be located at `<Location of MapLinkDirectImport DLL>/plugins/directimport/`. In a MapLink installation this corresponds to `<MapLink Bin Directory>/plugins/directimport/`.

As with other MapLink libraries there are multiple configurations of each driver. Only one of these configurations will be loaded at runtime based on the configuration of the MapLink Direct Import DLL.

15 TRACKS SDK

The TrackDisplayManager provides an easy way to create and display dynamic objects representing real-world entities that frequently change position. Tracks can be styled using application-defined symbols, or the APP6A and 2525B military symbology standards.

Each real-world entity is represented in the application by an instance of the Track class, which contains information about the entity such as its position, speed and velocity. Each track uses one or more symbol derived classes to define the appearance of the track at various zoom levels, as well as the appearance of the track's selection indicator (to visually identify when a track will be used for application-defined operations) and the appearance of the track's history trail.

Tracks are associated with a drawing surface using the TrackDisplayManager class. The TrackDisplayManager acts as a container for a group of tracks within the application, and provides methods for efficiently updating common properties on large numbers of tracks at once. It also provides the capability to record and replay the status of tracks over time, allowing the viewing of the history of the tracks in the TrackDisplayManager.

15.1 Library Usage and Configuration

As of version 11.1, MapLink is no longer supplied with Debug or 32-bit libraries. Therefore, your application's build should link against the Release Mode libraries in all configurations.

MapLinkTrackManager64.lib

Release mode, DLL version.

Uses Multithreaded DLL C++ run-time library.

Must also link the MapLink CoreSDK library MapLink.lib/MapLink64.lib

Requires TTLDLL preprocessor directive.

Refer to the document "MapLink Pro X.Y: Deployment of End User Applications" for a list of run-time dependencies when redistributing. Where X.Y is the version of MapLink you are deploying.

15.2 Track Display Manager Basics

See the API documentation for further details.

16 Dynamic Overlays with the DDO SDK

The Dynamic Data Object (DDO) SDK allows developers to create fully dynamic overlays within a MapLink application. Each object within this overlay can have application specific data associated with it through custom derivations of the base class. The architecture splits the real-world Data Object from the visualisation, allowing the same object to be displayed in different ways and in different positions according to application specific rules.

16.1 Library Usage and Configuration

As of version 11.1, MapLink is no longer supplied with Debug or 32-bit libraries. Therefore, your application's build should link against the Release Mode libraries in all configurations.

MapLinkDDO64.lib

Release mode, DLL version.

Uses Multithreaded DLL C++ run-time library.

Requires TTLDLL preprocessor directive.

Refer to the document "MapLink Pro X.Y: Deployment of End User Applications" for a list of run-time dependencies when redistributing, where X.Y is the version of MapLink you are deploying.

16.2 When to use Dynamic Data Objects

You have already seen how the Core SDK can be used to create dynamic overlays that are displayed in a `TSLStandardDataLayer`. These overlays are dynamically created, but typically change very little once they have been created. A Dynamic Data Object however is expected to be completely dynamic. There are several specific circumstances that would suggest that an Object Data Layer is used instead of a Standard Data Layer

- Objects are frequently created and destroyed.
- Objects are frequently moving or changing size.
- Completely different rendering is required on different Drawing Surfaces - e.g. displayed as a symbol in one surface and a polygon in another.
- Objects are displayed on multiple Drawing Surfaces with differing Coordinate Systems.
- Objects have significant amounts of application data.
- The rendering of an object requires the use of low-level Operating System calls for performance reasons, or for primitives that MapLink does not support.

16.3 Object Data Layers

The `TSLObjectDataLayer` class is a Data Layer, just like the `TSLMapDataLayer` and `TSLStandardDataLayer` that you have previously encountered. As such, it may be created and added to one or more Drawing Surfaces from whence the contents are displayed.

In the same way that a `TSLStandardDataLayer` contains instances of `TSLEntity` derived objects, the `TSLObjectDataLayer` contains instances of `TSLDynamicDataObject` derived objects. `TSLDynamicDataObject` is an abstract class and must be derived from before it can be used. Each Dynamic Data Object has

- A real-world position and extent.
- Optional, application-specific connection to a database or live feed.
- One or more visualisation objects - one for each Drawing Surface that the owning `TSLObjectDataLayer` is attached to.

The visualisation objects are instances of classes derived from the abstract `TSLDisplayObject`. In MapLink parlance, these are Display Objects. When an Object Data Layer is added to a Drawing Surface, all Dynamic Data Objects that it currently contains have their `instantiateDO` method called in order to create a Display Object for that Drawing Surface. When a new Dynamic Data Object is added to an Object Data Layer, the `instantiateDO` method is called for each Drawing Surface that the layer is currently attached to.

Query methods are available on the Object Data Layer to obtain a list of Dynamic Data Objects in the layer, the Display Objects associated with a particular Drawing Surface or the Display Objects within a particular area.

In addition to the position and extent associated with a Dynamic Data Object, Display Objects have their own position and extent. By default, these are identical to those of the owning Dynamic Data Object however they can be changed. This separation can be useful under several circumstances:

- When the Display Object can be dragged or moved away from the position of the Dynamic Data Object to prevent clutter or hiding of underlay data.
- When the Drawing Surfaces that the Object Data Layer is attached to have different coordinate systems or TMC coordinate spaces.
- When the Display Object is fixed-size in pixels and hence requires a different TMC extent in each Drawing Surface.

When the Object Data Layer is drawn onto a Drawing Surface, the list of Display Objects is iterated and any Display Objects whose extent overlaps the drawn extent have their ‘draw’ method called. This method is passed a pointer to a `TSLRenderingInterface` object which may be used to perform low-level rendering commands directly onto the Drawing Surface without the need for the creation of geometric Entities.

16.4 Custom Dynamic Data Objects

`TSLDynamicDataObject` is an abstract class[6] and must be derived from to be of use in an application. The key things to take note of are that the destructor of the derived class should be virtual and that the `instantiateDO` method should be implemented. The signature of this method is:

```
TSLDisplayObject* instantiateDO(TSLDisplayType key, int dsID = 0)
const ;
```

Note that this is a 'const' method[7]. A common mistake is to miss off the const modifier resulting in the derived method never being triggered and a run-time error generated. The key parameter is now obsolete and may be safely ignored. The `dsID` is the identifier of the Drawing Surface as specified with the `TSLDrawingSurface::id` method. It is supplied so that decisions can be made about the specific Drawing Surface that the Display Object is being instantiated for.

For simple applications, the only thing required is to maintain the position and extent of the Dynamic Data Object using the various position, move, translate and `setExtent` methods. In such applications, it is recommended that any `updateDOextent` parameters are set to true.

- The move methods set the position and extent of the Dynamic Data Object and optionally update the Display Objects positions and extents.
- The translate methods adjust the position and extent of the Dynamic Data Object by the specified delta values and optionally update the Display Objects positions and extents.
- The position methods affect the position of the Dynamic Data Object without updating any Display Objects or the Dynamic Data Object extent. This is usually only called during initialisation of the Dynamic Data Object.

The derived class may contain any application specific information required and may be driven by an application controlled external data feed.

16.5 Custom Display Objects

`TSLDisplayObject` is an abstract class and must be derived from to be of use in an application. The key things to take note of are that the destructor of the derived class should be virtual and that the draw method should be implemented. It is recommended,

[6] This isn't strictly true! For backwards compatibility, there is a base class implementation of the '`instantiateDO`' method. However, in practise, this should always be implemented by a derived class. If you forget to provide this in a new class, then a run-time error, `DDO_INSTANTIATEDO_NOT_OVERRIDDEN`, will be placed onto the error stack.

[7] An issue with the Rational Rose documentation generator means that the const modifier is not shown in the MapLink API documentation.

but not required, that a copy constructor is also implemented[8]. The signature of the draw method is:

```
bool draw(TSLRenderingInterface *ri, TSLEnvelope *extent);
```

A simple custom implementation of this method will make a sequence of calls to the Rendering Interface to set up attributes and draw graphical primitives. The Rendering Interface also provides facilities for coordinate conversion and access to the low-level `Drawable` or `HDC`. The varieties of attributes available are discussed in sections 10.6.1.

For simple applications, the Display Object position and extent will be the same as the owning Dynamic Data Object. More complex applications often involving multiple representations are discussed in section 16.7. Note that it is the Display Object extent that determines whether it is displayed, not the Dynamic Data Object extent. To modify the position and extent of the Display Object independently from its owner, the following methods may be used:

- The move methods set the position of the Display Object and update the extents accordingly.
- The translate methods adjust the position of Display Object by the specified delta values and update the extents accordingly.
- The position methods affect the position of the Display Object without updating the extent. This is usually only called during initialisation of the Display Object.
- The `setExtent` methods adjust the extent of the Display Object without affecting the position.
- The `setSize` method updates the extent of the Display Object by defining a size around the origin rather than the position. This is usually only called during initialisation of the Display Object when the position of the owning Dynamic Data Object will subsequently be used to update the extent.
- The `getExtent` method returns the current TMC extent of the Display Object. Note, for fixed pixel size Display Objects, the current zoom factor is used to convert the pixel size into a TMC extent.

For Display Objects that are fixed size in Device Units rather than in TMC Units, the following methods may be used:

- The `setPixSize` method defines the size of the Display Object with a pixel extent around the origin. This is used dynamically during the rendering pass to determine whether the Display Object should be drawn. To override this once it has been set use the normal `setSize`, or `setExtent` methods.
- The `getPixSize` method queries the extent that was defined using `setPixSize`. Note, for Display Objects that are not fixed size in Device Units, this method returns current TMC extent.

[8] This is mainly used by the obsolete `clone` method. The `clone` and `unclone` methods are for backwards compatibility only and do not need to be overridden.

- The `fixedPixSize` method can be used to query whether the Display Object is fixed size in Device Units.

16.6 Walkthrough 4 – Adding Simple Dynamic Objects

This section guides you through adding a simple Dynamic Object Layer to the MapLink application that has been developed in the earlier walkthroughs. By the end, you should have an application that displays a single object which tracks the mouse cursor as it moves over the map.

The example is based on MFC and the C++ SDK, but the same steps apply on X11 targets and with the other MapLink SDK's.

16.6.1 Configure Project Settings

You need to set up the project settings according to the version of the MapLink libraries you wish to use. These are described in section 16.1 and must match those used for the Core MapLink SDK.

Check/change the following settings in Project Properties:

x64 configuration: check/change the following settings in Project Properties:

Under the Link,Input category, add `MapLinkDDO64.lib` as an object/library.

Add `#include "MapLinkDDO.h"` to relevant files. In this example, just add it into `stdafx.h` to keep things simple.

16.6.2 Adding a TSLObjectDataLayer

The Object Data Layer will be used to manage the Dynamic Data Object and will need to be added to the document class along with the other Data Layers. It should be created and destroyed where appropriate and added to the `TSLDrawingSurface` when the document and view are bound together. As an optimisation, we will also make the Map Data Layer double buffered to avoid having to redraw it every time the Dynamic Objects are updated.

In the Document class definition, add a declaration of the Object Data Layer just after the Standard Data Layer:

```
TSLMapDataLayer      * m_mapDataLayer ;
TSLStandardDataLayer * m_stdDataLayer ;
TSLObjectDataLayer   * m_objDataLayer ; // This line added
```

The new class variable should be initialised to 0 in the Document constructor.

```
CHelloGlobeDoc::CHelloGlobeDoc()
 : m_mapDataLayer( NULL ),
 m_stdDataLayer( NULL ),
 m_objDataLayer( NULL )
```

The layer should only be created after a map has been successfully loaded and should be destroyed when the map layer is destroyed.

In the Document `OnOpenDocument` method, instantiate a `TSLObjectDataLayer` if the map is successful:

```
if ( !m_mapDataLayer->loadData( lpszPathName ) )
{
  // Error handling as before
  return FALSE ;
}
m_stdDataLayer = new TSLStandardDataLayer() ;
m_objDataLayer = new TSLObjectDataLayer() ;
```

In the Document `DeleteContents` method, add the following code to delete the overlay layer:

```
if ( m_objDataLayer )
{
  m_objDataLayer->destroy() ;
  m_objDataLayer = NULL ;
}
```

Modify the Document `addToSurface` method as below to add the extra layer and to make the map layer buffered:

```
if (   !m_mapDataLayer || !m_stdDataLayer
    || !m_objDatalayer || !drawingSurface )
{
  return false ;
}
bool sts = drawingSurface->addDataLayer( m_mapDataLayer, "map" ) ;
if ( sts )
{
  drawingSurface->setDataLayerProps("map",TSLPropertyBuffered,true);
  sts = drawingSurface->addDataLayer( m_stdDataLayer, "overlay" ) ;
}

if ( sts )
  sts = drawingSurface->addDataLayer( m_objDataLayer, "dynamic" ) ;

return sts ;
```

16.6.3 Creating a Custom Dynamic Data Object

Two custom classes are required – one derived from `TSLDynamicDataObject` to supply the interface to the application data and one derived from `TSLDisplayObject` to provide the visualisation.

Create a new class, MyDDO which derives from TSLDynamicDataObject and override the instantiateDO method. Include the header file in the Document source file.

```
class MyDDO : public TSLDynamicDataObject
{
  public:
    MyDDO(void);
    virtual ~MyDDO(void);
    virtual TSLDisplayObject * instantiateDO(TSLDisplayType key, int dsID=0) const ;
};
```

Create a new class, MyDO which derives from TSLDisplayObject and override the draw method. Include the header file in the MyDDO source file.

```
class MyDO : public TSLDisplayObject
{
  public:
    MyDO(void);
    virtual ~MyDO(void);
    virtual bool draw(TSLRenderingInterface *ri, TSLEnvelope *extent);
}
```

In the source file for MyDDO, provide initial empty definitions for the constructor, and destructor and a simple implementation for the instantiateDo method.

```
MyDDO::MyDDO(void)    {  }
MyDDO::~MyDDO(void)   {  }
TSLDisplayObject * MyDDO::instantiateDO(TSLDisplayType key,int dsID) const
{
  return new MyDO() ;
}
```

In the source file for MyDO, provide initial definitions for the constructor, and destructor and a simple implementation for the draw method – draw a red circle 50 pixels high.

```
MyDO::MyDO(void)
{  // Without this, the symbol disappears when the centre goes off screen
   setPixSize( -25, -25, 25, 25 ) ;
}
MyDO::~MyDO(void)  {  }
bool MyDO::draw( TSLRenderingInterface *ri, TSLEnvelope *extent )
{ // 1023 = filled circle with cross, 181=red in standard config files
  ri->setupSymbolAttributes( 1023, 181, 50, TSLDimensionUnitsPixels );
  ri->drawSymbol( position() ) ;
  return true ;
}
```

16.6.4 Moving the Dynamic Data Object

If you compile and run the application, then load the sample World map, you will see an object appear in the middle of the map. This is the Display Object that has been rendered using the `MyDO::draw` method. In a real application, there are likely to be many more objects with their positions being driven via some external data feed. For this example, however, we will merely update the position of the DDO on a mouse move event. This will cause updates in all views displaying the Object Data Layer so if you have created a multiple-document application then the object will move in all views attached to the document.

In the Document class definition, add a new public method `updateDDOPosition`, with the following definition:

```
bool CHelloGlobeDoc::updateDDOPosition( long x, long y )
{
  if ( m_objDataLayer )
  {
    // Get the solitary DDO - could iterate over DDO list
    TSLDynamicDataObject * ddo = m_objDataLayer->getDDO( 0 ) ;
    if ( ddo )
    {
      ddo->move( x, y, true ) ;          // true means also updates DO
      m_objDataLayer->notifyChanged() ; // Invalidates buffer
      UpdateAllViews( 0 ) ;             // Update views displaying doc
      return true ;
    }
  }
  return false ;
}
```

In the View class, modify the `OnMouseMove` handler to call this new method that should make the DDO track the mouse cursor. In the handler, declare a boolean variable '`moved`' at the top of the function and initialize it to `false`. If the call to '`pan`' is successful, then set this variable to '`true`' as well as invalidating the view rectangle. Add the following code before the call to `CView::OnMouseMove`

```
if ( m_drawingSurface && !moved )
{
  CHelloGlobeDoc * doc = GetDocument() ;

  TSLTMC x, y ;
  if ( m_drawingSurface->DUToTMC( point.x, point.y, &x, &y ) )
    doc->updateDDOPosition( x, y ) ;
}
```

Now compile and run your application. Load a map and you should see the DDO track the mouse across the window. If you have a multiple document application, create a new window on the document and note that both views are updated.

16.7 Advanced Uses of the Dynamic Data Object SDK

The earlier walkthrough has shown how to implement a simple dynamic overlay. Real world applications are rarely that simple however! Common issues are discussed in the following sections.

16.7.1 Multiple Representations

Many applications require an object to have different representations on different Drawing Surfaces. The Dynamic Data Object SDK allows you to implement these by instantiating a different Display Object on different Drawing Surfaces.

The visualisation is encapsulated within the Display Object and may thus be very specific to a particular usage. The Dynamic Data Object can distinguish between Drawing Surfaces by using the `dsID` parameter that is passed to the `instantiateDO` method. This is the value that has been set by the application using the `TSLDrawingSurface::id` method.

16.7.2 Multiple Coordinate Systems

Some classes of application have a common dynamic overlay displayed on top of multiple maps. These maps may be in different coordinate systems. For example, tracks may be displayed on a Mercator or Dynamic Arc overview map and on a zoomed-in detailed map – usually in an appropriate projection for the location such as the local UTM zone.

The Dynamic Data Object SDK allows you to set the position and extent of Display Objects independently of the Dynamic Data Objects. One way to make use of this feature is to hold the generic position of the real-world object within the Dynamic Data Object as latitude/longitude. During the `instantiateDO` call, overwrite the position of the Display Object with the TMC unit position appropriate to the Drawing Surface for which the Display Object is being created. A common trick is to set the `TSLDrawingSurface::id` to the address of the Drawing Surface, thus allowing the `TSLDrawingSurface` pointer to be used to perform coordinate transformations in the `instantiateDO` call.

If the Display Object position is updated in sympathy with the Drawing Surface coordinate system, then Display Objects can be positioned correctly in all circumstances.

16.7.3 Rendering using Xlib or Win32

For high performance, the low-level handle to the `Drawable` or `HDC` can be queried during rendering using the Rendering Interface `handleToDrawable` method. The application can then make low-level calls to create the visualisation. Note that the application must be careful to leave the low-level handle in the state it was when returned from `handleToDrawable` – for example, using the Win32 `saveDC` and `restoreDC` methods.

17 TERRAIN SDK

The Terrain SDK is available for any mapping application that requires the addition of the third dimension. Applications can use the data provided by the terrain SDK to perform height queries, visibility calculations or even render the map data in three dimensions.

17.1 Library Usage and Configuration

As of version 11.1, MapLink is no longer supplied with Debug or 32-bit libraries. Therefore, your application's build should link against the Release Mode libraries in all configurations.

TTLTerrain64.lib

Release mode, DLL version.

Uses Multithreaded DLL C++ run-time library.

Must also link the MapLink CoreSDK library MapLink.lib/MapLink64.lib

Requires TTLDLL preprocessor directive.

Refer to the document "MapLink Pro X.Y: Deployment of End User Applications" for a list of run-time dependencies when redistributing, where X.Y is the version of MapLink you are deploying.

17.2 Where to Begin?

The Terrain SDK has been designed with the same philosophy in mind as the rest of MapLink. That is, it can easily be integrated within any application with the minimum of fuss. Therefore, the SDK has been developed to be very easy to use providing tools to allow quick, easy access to the data.

The design of the Terrain SDK means that it is very simple to use. The process required to access Terrain data within an application is as simple as the two steps below:

- Create an instance of the Terrain SDK main class object.
- Load some terrain data into the object.

Add the following to your application to create an instance of the Terrain SDK main class and load some data into it:

```
TSLTerrainDatabase* terrainDB = new TSLTerrainDatabase;
if ( terrainDB->open( "terraindb.tdb" ) != TSLTerrain_OK )
{
  // Handle file open error
}
else
{
  // Terrain database is open and ready to query. Start by getting
  // the extent of the database
  double x1, y1, x2, y2;
  terrainDB->queryExtent( x1, y1, x2, y2 );
}
```

Once you have performed these two steps, you are ready to start querying the data.

Notice in the code fragment above that we have used the `queryExtent` function to determine the coverage of the terrain database. This function returns coordinates in Map Units (see 17.4) that define the bounding box for the terrain database.

The function, `TSLTerrainDatabase::open` takes an optional second parameter – a pointer to a `TSLPathList`. If specified, the Terrain SDK looks for the file using the path list. See the online documentation on `TSLPathList` for further information.

17.3 How Fast is Fast?

The architecture of the Terrain SDK has been optimised to allow efficient queries of height data over any area extent. Within an application you can switch from performing queries covering the whole world to performing queries within 1 square kilometre without any significant change in performance or quality.

17.3.1 How does this work?

The Terrain SDK takes advantage of the fact that most queries on the Terrain database are used to generate an output that is displayed to the user. For example, the Terrain data might be used to display the height cross-section between two points. The resolution of this data is limited by the resolution of the screen display; therefore, it is not worth querying more points than can be represented on the screen. This is compounded by the fact that the cross-section query might be generated from the user dragging a line between the two points on a map display. This is also obviously limited by the resolution of the screen display.

The Terrain database prepared by MapLink Studio is tiled at multiple resolutions. This allows the Terrain SDK to choose the correct resolution depending on the type of query. For example, a query covering the whole world will use a low-resolution layer whereas a query covering a small area will use a higher resolution layer.

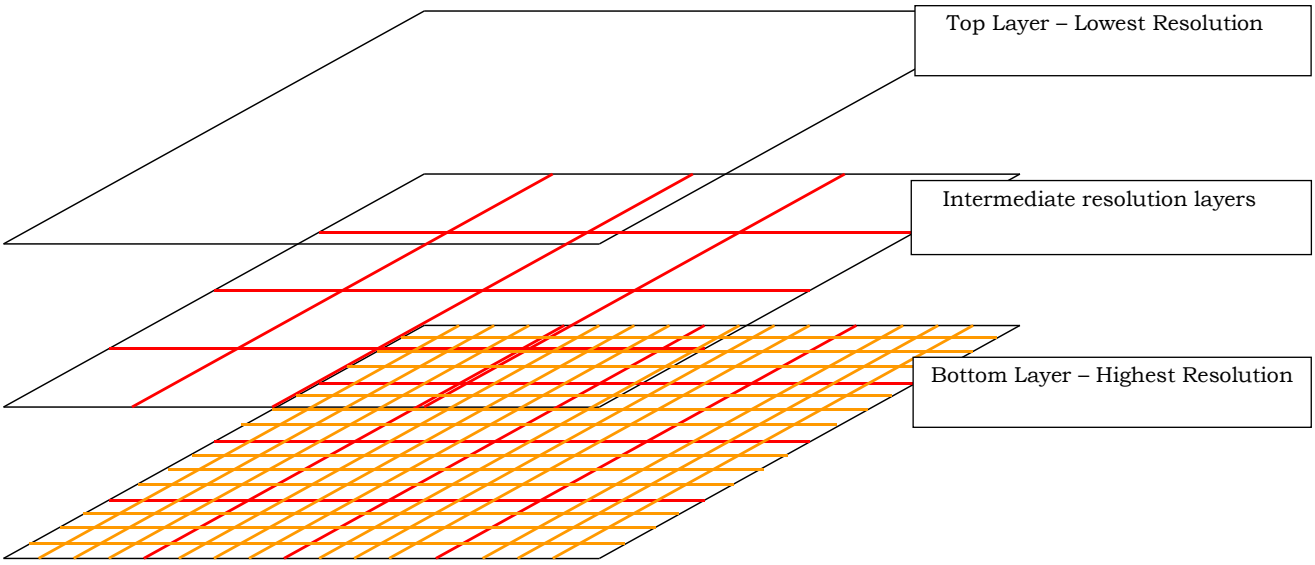


Figure 23 Terrain Pyramid

The Terrain SDK must be given enough information to determine the optimal level to use. Whenever the extent of the area of terrain data you are interested in changes, you must tell the Terrain SDK of the new extent. The Terrain SDK will then reconfigure itself for the new extent. If the area of interest changes but the extent remains the same (i.e. during a pan operation) then it is not necessary to notify the Terrain SDK.

17.4 Lining it All Up (Coordinate Systems)

When a map is prepared using MapLink Studio, the map is generated in a specific coordinate system defined by the MapLink Studio project. The same is also true for Terrain Databases generated using MapLink Studio. Therefore, when working with the Terrain SDK it is important to know what coordinate system the Terrain Database uses.

All functions within the Terrain SDK that take coordinates as a parameter expect the coordinates to be in Map Units (MU). Map Units are defined by the Output Coordinate System defined in the MapLink Studio project. They are generally in metres, but this does not always have to be the case. For example, if a terrain database is generated using the MapLink "Default Coordinate System" i.e. no coordinate system is configured; the Map Units will be in plain old WGS84 Latitude/Longitude positions. Whereas if the output coordinate system is configured as UTM Zone 30 North, the Map Units will be in metres with the origin being the centre of the projection.

The Terrain SDK provides functions to allow the coordinate system to be queried as well as allowing conversion between MU and latitude/longitude. If you want to check if the currently loaded terrain database is in a coordinate system that your application can handle, then add the following code after the call to open the terrain database.

Add this code after the call to `terrainDB->open()` to ensure the loaded terrain database is using the default coordinate system:

```
TSLCoordinateSystem* cs = terrainDB->queryCoordinateSystem();
// In most cases cs will not be NULL but older versions of the
// Terrain Database did not support coordinate system queries
// therefore we must check the return value of
// queryCoordinateSystem()
if ( cs )
{
  // Make sure the coordinate system is the default coordinate
  // system so we can assume the Map Units are in Latitude/Longitude
  if ( stricmp( cs->name(), "Default Coordinate System" ) != 0 )
  {
  // Not the default coordinate system - display an error
  }
}
```

The `TSLCoordinateSystem` object returned by `TSLTerrainDatabase::queryCoordinateSystem` can be used to convert from MU to Latitude/Longitude and vice versa. In addition, the same methods have been provided in the API interface to `TSLTerrainDatabase`. The `TSLTerrainDatabase` methods are functionally identical to the `TSLCoordinateSystem` methods and are provided purely for convenience.

The first method for transforming coordinates uses `TSLCoordinateSystem`:

```
TSLCoordinateSystem* cs = terrainDB->queryCoordinateSystem();
if ( cs )
{
  double muOutX, muOutY;
  double latOut, lonOut;
  if ( cs->latLongToMU( latIn, lonIn, &muOutX, &muOutY ) )
  {
    // Conversion successful
  }
  if ( cs->MUToLatLong( muInX, muInY, &latOut, &lonOut ) )
  {
    // Conversion successful
  }
}
```

The second method for transforming coordinates is more convenient:

```
double muOutX, muOutY;
double latOut, lonOut;
if ( terrainDB->latLongToMU( latIn, lonIn, &muOutX, &muOutY ) ==
                                                    TSLTerrain_OK )
{
  // Conversion successful
}
if ( terrainDB->MUToLatLong( muInX, muInY, &latOut, &lonOut ) ==
                                                    TSLTerrain_OK )
{
  // Conversion successful
}
```

17.5 How Do I Access the Data?

Once a terrain database has been opened, querying the data is simple. There are three methods provided which allow the database to be queried. The choice of query function you use is dependent on your application – choose the function that is most convenient. Each of the query functions returns whether the query was successful or not. Possible return values for the query functions are:

`TSLTerrain_OK`	The query was successful.
`TSLTerrain_NoData`	There was no data in the database for the requested position.
`TSLTerrain_???`	Any other error conditions

To query a line of 10 points from the database:

```
TSLTerrainDataItem dataItem[10];
if ( terrainDB->queryLine( muXstart, muYstart, muXend, muYend,
                                    10, dataItem ) == TSLTerrain_OK )
{
  // Query successful. The information about the point is
  // stored in the dataItem array
}
```

To query a single point from the database:

```
TSLTerrainDataItem dataItem;
if ( terrainDB->query( muX, muY, &dataItem ) == TSLTerrain_OK )
{
  // Query successful. The information about the point is
  // stored in dataItem
}
```

To query a 10 x 5 grid from the database:

```
TSLTerrainDataItem dataItem[10*5];
if ( terrainDB->queryArea( muBlX, muBlY, muTrX, muTrY,
                           10, 5, dataItem ) == TSLTerrain_OK )
{
  // Query successful. The information about the point is
  // stored in the dataItem array. The data is stored row-by-row
}
```

In each case, the data is returned in one or more `TSLTerrainDataItem` objects. For efficiency, by default the query functions will only populate the fields that define the requested position and the height/depth. It is important to note that even if the query function returns `TSLTerrain_OK`, the height value may not be valid. This is because some databases may contain 'holes' in their data coverage. This is indicated by the `TSLTerrainDataItem::m_isNull` flag being set – see the table below.

You may have noticed that one of the optional parameters to the query functions is a "filter". This is used to define which fields to populate. The data returned by each filter is defined in the table below:

Field	Filter	Description
`m_x`	`TSLTerrainData_Min`	MU X position of queried data
`m_y`	`TSLTerrainData_Min`	MU Y position of queried data
`m_z`	`TSLTerrainData_Min`	Height/Depth of requested position in metres
`m_isNull`	`TSLTerrainData_Min`	Flag that is set to true if no data existed at the requested position. If set to false, the `m_z` member contains valid data
`m_nearestX`	`TSLTerrainData_Nearest`	Contains the MU x position of the nearest actual point within the terrain database
`m_nearestY`	`TSLTerrainData_Nearest`	Contains the MU y position of the nearest actual point within the terrain database
`m_nearestZ`	`TSLTerrainData_Nearest`	Contains the height value for the point in the database at `m_nearestX`, `m_nearestY`. This field is only valid if `m_nearestIsNull` is false
`m_nearestIsNull`	`TSLTerrainData_Nearest`	If true, `m_nearestZ` is not valid otherwise it is valid

Field	Filter	Description
m_xResolution	TSLTerrainData_HorizontalRes	Defines the spacing of columns in the grid. The value is in MU.
m_yResolution	TSLTerrainData_HorizontalRes	Defines the spacing of rows in the grid. The value is in MU.

Some of the fields in TSLTerrainDataItem are unused in the current implementation of the Terrain SDK. Only the fields in the table above are currently used.

Any number of filters can be combined to retrieve the required information. TSLTerrainData_Min is always added regardless of what other filter flags are set. A convenient definition is provided to get all available data: TSLTerrainData_All.

17.6 What Happens When There Is No Data for a Point? (Interpolation)

The simple answer is for the situation when a point is requested outside the extent returned by a call to TSLTerrainDatabase::queryExtent. In this case, the function just returns TSLTerrain_NoData. No further information is available. If, however, the point lies within the extent but between actual entries in the database rather than directly on an entry, a value is returned. The Terrain SDK calculates this value depending on the interpolation parameter passed into the query function.

Available values for the interpolation parameter are:

TSLTerrainInterpolate_NONE	A nearest neighbour algorithm is used that returns the height of the point nearest the requested point. Very fast but less accurate.
TSLTerrainInterpolate_LINEAR	Bilinear interpolation is used to calculate height value. More accurate but slightly slower.
TSLTerrainInterpolate_MIN, TSLTerrainInterpolate_MAX	The lowest/highest neighbouring value is used. Very fast but rarely used due to their inaccuracy.

Whichever interpolation value is used, the filter parameter can be used to determine the position and height value of the nearest point to the requested location. This can be useful when for reasons of accuracy; only actual values stored within the database are to be used in an application. This is illustrated in the code below:

To query a single point from the database getting all available associated information:

```
TSLTerrainDataItem dataItem;
if (   terrainDB->query( muX, muY, &dataItem, TSLTerrainData_All )
    == TSLTerrain_OK )
{
  // Query successful. All information about the point is
  // stored in dataItem
}
```

17.7 How Accurate is My Data? (Querying Different Levels)

Referring back to section 17.3, we remember that the Terrain Database is stored in a pyramid structure to allow very quick queries of the data. How does the Terrain SDK know which pyramid level we want to use? In some cases, we want to obtain the most accurate height value for a location. How do we make sure we use the most detailed layer?

The second question in the paragraph above is the easiest to answer; "How do we make sure we use the most detailed layer?" There is one more optional parameter for the query functions that we haven't looked at yet. Setting the Boolean parameter, `highestRes` to true notifies the Terrain SDK that we want the height value to be read from the highest resolution layer that contains the requested point. Setting it to false will allow the Terrain SDK to optimise the speed of the request at the expense of some of the accuracy.

Beware when using the `highestRes` parameter when covering a large area of the terrain database – the query could take a substantial amount of time.

To query a single point from the database using the highest resolution layer available. Note the "true" parameter to the query function that informs the Terrain SDK that we want the highest resolution data.

```
TSLTerrainDataItem dataItem;
if (terrainDB->query(muX, muY, &dataItem, TSLTerrainData_Min, true)
                                                    == TSLTerrain_OK )
{
  // Query successful. Highest resolution data obtained
}
```

To query a single point from the database getting all available associated information:

```
TSLTerrainDataItem dataItem;
// Use bilinear interpolation for a more accurate result
if ( terrainDB->query( muX, muY, &dataItem,
                  TSLTerrainData_Min | TSLTerrainData_Nearest, false,
                  TSLTerrainInterpolate_LINEAR ) == TSLTerrain_OK )
{
  // Query successful.
  // m_x and m_y contain the requested location.
  // m_z contains a value obtained by bilinear interpolation
  // m_nearestX, m_nearestY and m_nearestZ contain the location
  // and height/depth of the nearest stored data in the database
}
```

Onto the other question: "How does the Terrain SDK know which layer in the pyramid to use?" The simple answer is: you have to tell it! The mechanism for telling the Terrain SDK what resolution of data you want has been designed to tie-in with a MapLink application that displays map data. A quick explanation is required:

In most MapLink applications, a map display is provided and the user is given controls to pan, zoom, etc. Quite often, the Terrain SDK is incorporated into the application to allow the user to perform queries on terrain data. For example, the application can display a cross-section of the terrain between two points when the user drags a line over the map display. You can see from this example that it is pointless querying more points from the terrain database than can be displayed by the resolution of the screen. The Terrain SDK takes advantage of this and adjusts the pyramid level accordingly.

This is surprisingly simple to setup within a MapLink application – all you need to do is tell the Terrain SDK whenever the map extent changes – i.e. on a zoom operation or when the map window size changes. Note that it is not necessary to tell the Terrain SDK when the map is panned as this does not change the extent of the map.

In the handler for the MapLink map zoom command, notify the Terrain SDK of the new extent.

```
if ( m_drawingSurface->zoom( 25, true, false ) )
{
  // Zoom was successful. Tell the Terrain SDK
  // The function that notifies the Terrain SDK requires the size
  // of the map window and the new extent. Assume the map window
  // is the same size as this window
  CRect rc;
  GetClientRect( rc );

  // Get the extent of the data. We need to convert this to MU. This
  // assumes the MU of the map and of the Terrain SDK are the same
  double muX1, muY1, muX2, muY2;
  if ( m_drawingSurface->getMUExtent( &muX1, &muY1, &muX2, &muY2 ) )
  {
    // We have enough information now
    terrainDB->displayExtent( rc.Width(), rc.Height(), muX1, muY1,
                                                        muX2, muY2 );
    // That's it! Terrain database optimised for this
    // screen resolution
  }
  InvalidateRect( 0, FALSE ) ;
```

17.8 Contouring

The Terrain SDK also allows for the generation of contour lines or polygons from the same height information used in a terrain database. The format that the generated contour information is displayed in is controlled entirely by the application via the use of rendering callbacks.

17.8.1 Providing Data for Contouring

The data to contour is expected in the form of a `TSLTerrainContourVertexList` of `TSLTerrainContourVertex` objects. Each vertex object represents data at a single point, and when all vertices are combined, they should form a regular or irregular grid inside the list object.

Each vertex can store one or more pieces of height information, named 'attributes', for the point it represents. Each of these attributes can be used to model different information about the point that the vertex represents. For example, the first attribute might be height information for the terrain at that point, a second attribute might be a recorded temperature value at that point and a third attribute might be a humidity value. Contour information can be generated separately for each of these attributes. Each vertex within the list must have the same number of attributes.

This example shows loading of height information from a terrain database and storing the data in a `TSLTerrainContourVertexList` ready for the generation of contour lines.

```
// Process the terrain data into a terrain database
if( m_terrainDB.open( terrainDBFile.c_str() ) != TSLTerrain_OK )
  return false;

// Query the extent of the terrain data
long x1, y1, x2, y2;
if( m_terrainDB.queryExtent( x1, y1, x2, y2 ) != TSLTerrain_OK )
  return false;

// Inform the terrain database of the size of our drawing surface
// so it can determine a good resolution for the data
long duMinX, duMaxX, duMinY, duMaxY;
m_drawingSurface->getDUExtent( &duMinX, &duMinY, &duMaxX, &duMaxY );
m_terrainDB.displayExtent( duMaxX - duMinX, duMaxY - duMinY,
   x1, y1, x2, y2 );

// Read the data from the terrain database
TSLTerrainDataItem *dataItems =
  new TSLTerrainDataItem[ m_terrainGridWidth * m_terrainGridHeight ];
if( m_terrainDB.queryArea( x1, y1, x2, y2, m_terrainGridWidth,
                           m_terrainGridHeight,
                           dataItems ) != TSLTerrain_OK )
{
  return false;
}

// Convert the terrain database to contour vertices so we can give
// them to the contour object
TSLTerrainContourVertexList *vertices =
                                    new TSLTerrainContourVertexList();

for( int i = 0; i < m_terrainGridHeight; ++i )
{
  for( int j = 0; j < m_terrainGridWidth; j++ )
  {
    vertices->addVertex(dataItems[(i * m_terrainGridWidth ) + j].m_x,
                        dataItems[(i * m_terrainGridWidth ) + j].m_y,
                        1,
                        &dataItems[(i * m_terrainGridWidth) + j].m_z);
  }
}

// Height information is now stored in the vertex list so the data
// from the terrain database is no longer required
delete[] dataItems;

TSLTerrainContour contour = new TSLTerrainContour();

// Give our vertex list to the contour object so we can then perform
// contouring - the contour object assumes ownership of the vertex
// list
contour->setVertices( vertices );
```

Although the contour object assumes ownership of the vertex list, the data contained within the list can still be modified by the application without having to generate a new vertex list and setting it on the contour object. This avoids having to do large copies when you wish to modify the data used for contouring. If this is done, the `TSLTerrainContour` object should be informed of the change via the `notifyChanged()` method in order to ensure that the updated data is used for future contouring operations.

17.8.2 Types of Contours

Contour information can be generated either as polygons or lines. When generating contours as lines there are three different algorithms that can be used, specified by the `TSLTerrainContourLineType` enumeration. The simplest of these is `TSLTerrainContourLineTypeSimple` which uses a Triangulated Irregular Network (TIN) to calculate the contour lines. `TSLTerrainContourLineTypeStandard` uses a similar method but performs some optimisation on the resulting contour lines to remove duplicate points from the calculated contours. `TSLTerrainContourLineTypeCONREC` uses a different algorithm that in most cases produces contour lines as good as those generated by the simple or standard methods but is substantially faster.

When generating contours as polygons there is no algorithm choice to make.

17.8.3 Drawing the Contours

Contours generated from the `TSLTerrainContour` class are passed to the application via one of the `TSLTerrainContourCallbacks` virtual methods. Which callback is invoked is dependent on which type of contour (see section 17.8.2) was requested according to the following table:

Callback	Used by
`TSLTerrainContourCallbacks::progress`	All

`TSLTerrainContourCallbacks::drawLine`	`TSLTerrainContour::drawContourLine` using the following types:
	`TSLTerrainContourLineTypeSimple`
	`TSLTerrainContourLineTypeCONREC`

`TSLTerrainContourCallbacks::drawPolygon`	`TSLTerrainContour::drawContourPolygon`
`TSLTerrainContourCallbacks::drawPolyline`	`TSLTerrainContour::drawContourLine` using the following types: `TSLTerrainContourLineTypeStandard`
`TSLTerrainContourCallbacks::drawText`	`TSLTerrainContour::drawContourLine` using the following types: `TSLTerrainContourLineTypeStandard`
`TSLTerrainContourCallbacks::drawTIN`	`TSLTerrainContour::drawTIN`

You should override each of the callbacks that will be used for your selected method of contour generation. The `TSLTerrainContourCallbacks` class provides default

implementations of all the callbacks so that you only need to implement the ones that you are interested in.

The callbacks will be invoked numerous times before the original draw call returns. In order to prevent excessive redrawing your application should wait until the draw call has returned before updating the display of your application.

This example shows an implementation of the

`TSLTerrainContourCallbacks::drawPolyline()` callback in which the generated contour lines are added to a `TSLStandardDataLayer` to be drawn to the screen after contour generation has finished.

```
void TerrainContouringView::drawPolyline (TSLTerrainContourVertexList*
vertices, double attribute)
{
  // The coordinates of the vertices given to us are in the coordinate
  // system of the terrain data, which may not be the same as that of
  // the map we have loaded. Therefore it may be necessary to
  // convert the coordinates so the contour lines appear in the correct
  // place on the map.
  TSLCoordinateSystem *terrainCS =
                              m_terrainDatabase->queryCoordinateSystem();
  const TSLCoordinateSystem *mapCS =
                                 m_mapDataLayer->queryCoordinateSystem();
  bool needToConvert = false;
  if( terrainCS->id() != mapCS->id() ||
      terrainCS->getTMCperMU() != mapCS->getTMCperMU() )
    needToConvert = true;

  TSLCoordSet *coords = new TSLCoordSet();

  // Process the list of vertices given to us into a polyline so we can
  // display it on the map in a standard data layer
  for( int i = 0; i < vertices->numberOfVertices(); ++i )
  {
    TSLTerrainContourVertex &vertex = vertices->at(i);
    TSLTMC tmcX = 0, tmcY = 0;

    if( !needToConvert )
    {
      // The terrain database and map coordinate systems are the same
      terrainCS->MUToTMC( vertex.x(), vertex.y(), &tmcX, &tmcY );
    }
    else
    {
      // Convert between the terrain database and map coordinate systems
      double lat = 0.0, lon = 0.0;
      terrainCS->MUToLatLong( vertex.x(), vertex.y(), &lat, &lon );
      mapCS->latLongToTMC( lat, lon, &tmcX, &tmcY );
    }

    coords->add( tmcX, tmcY );
  }

  TSLEntitySet *es = m_contourLayer->entitySet();
  TSLPolyline *line = es->createPolyline( 0, coords, true );
  if( line )
  {
    line->setRendering( TSLRenderingAttributeEdgeStyle, 1 ) ;

    // Determine line colour based on the height of the contour
    long colour = ( 255 / m_maxTerrainHeight ) * attribute;
    line->setRendering( TSLRenderingAttributeEdgeColour,
    TSLDrawingSurface::getIDOfNearestColour( colour, 0, 255 - colour) );
    line->setRendering( TSLRenderingAttributeEdgeThickness, 1 ) ;
  }
}
```

17.8.4 Drawing the Contour Labels

When drawing contour lines using `TSLTerrainContourLineTypeStandard` there is the option to draw labels for the generated contour lines. This is enabled by passing a non-NULL value to the `textPrefix` parameter of the `TSLTerrainContour::drawContourLine()` method, which will be passed to the `TSLTerrainContourCallbacks::drawText()` callback. This is usually set to a description of what the value in the label will represent (e.g. 'Height:' or 'Temperature:'), but if nothing is desired can be set to an empty string.

When using text labels with the alignment value set to `TSLVerticalAlignmentMiddle` the contour lines are split at appropriate points around the labels so that the lines do not run through the labels themselves. As the contents of the labels are controlled via the application by the `TSLTerrainContourCallbacks::drawText()`, this necessitates informing the contour object of the maximum length that the text strings will be when the `TSLTerrainContour::drawContourLine()` method is invoked.

One way of doing this is to create a dummy text object of the longest expected length and use this to determine the size to pass in as follows:

```
TSLText *textObj = m_contourLayer->entitySet()->createText( 0, 0, 0,
                                    maxLengthLabel.str().c_str(), 100 );

// It is necessary to set up the following attributes on the text object
// for updateEntityExtent() to work
textObj->setRendering( TSLRenderingAttributeTextSizeFactor,
                       m_textSizeFactor );
textObj->setRendering( TSLRenderingAttributeTextSizeFactorUnits,
                       TSLDimensionUnitsMapUnits );
textObj->setRendering( TSLRenderingAttributeTextFont, 2 );

// Set an entity ID on the temporary text object so we can remove it
// once we're done
textObj->entityID( INT_MAX );

m_contourLayer->notifyChanged();

// Store the currently viewed area of the map. In order to calculate the
// extent of the text object we need to change the viewed area so that
// our temporary text object would be visible
double viewedUUX1, viewedUUY1, viewedUUX2, viewedUUY2;
m_drawingSurface->getUUExtent( &viewedUUX1, &viewedUUY1,
                               &viewedUUX2, &viewedUUY2 );

double newUUX1, newUUY1, newUUX2, newUUY2;
long newSizeArea = 2 * m_textSizeFactor;
m_drawingSurface->MUToUU( -newSizeArea, -newSizeArea,
                          &newUUX1, &newUUY1 );
m_drawingSurface->MUToUU( newSizeArea, newSizeArea,
                          &newUUX2, &newUUY2 );
m_drawingSurface->resize( newUUX1, newUUY1,
                          newUUX2, newUUY2, false, true );

// Now calculate the size of our text object
m_drawingSurface->updateEntityExtent( textObj );
TSLEnvelope env = textObj->envelope( m_drawingSurface->id() );
unsigned long envWidth = env.width();

// Now we have the width of the text object in TMCs we need to convert this to
the terrain database units
double lat1, lon1, lat2, lon2, x1, y1, x2, y2;
m_drawingSurface->TMCToLatLong( env.bottomLeft().x(),
                                env.bottomLeft().y(), &lat1, &lon1 );
m_drawingSurface->TMCToLatLong( env.topRight().x(),
                                env.topRight().y(), &lat2, &lon2 );
m_terrainDB.latLongToMU( lat1, lon1, &x1, &y1 );
m_terrainDB.latLongToMU( lat2, lon2, &x2, &y2 );

// This is the width of the text labels in the terrain database units
// with some additional space either side
width = ( x2 - x1 ) * 1.5;

// Now we have the width we no longer need our text object
m_contourLayer->removeEntity( INT_MAX );

// Finally, reset the viewied area of the map back to what it was originally
m_drawingSurface->resize( viewedUUX1, viewedUUY1,
                          viewedUUX2, viewedUUY2, false, false );
```

17.8.5 Performance Notes

Calculating contours can take a considerable amount of time when given large amounts of data to work on. As the result of a draw operation will not change if the data remains the same, it is more sensible to store the results of the contouring operation in a form that

allows for fast rendering. The example in section 17.8.3 does this by creating geometry objects for each contouring line and storing them in a `TSLStandardDataLayer`. This prevents needless recalculation of the same points on each draw in the application.

17.9 Intervisibility/Viewshed Calculations

The Terrain SDK provides the ability to perform point-to-point line of sight calculations, along with area viewshed/intervisibility calculations. An area viewshed determines which points can/cannot be seen from a given start point by performing multiple line of sight calculations.

This functionality is exposed via a flexible set of classes, allowing the calculation to be integrated with a variety of applications. This API is currently only provided via the C++ interfaces.

The viewshed API consists of the following object types:

- Input objects
- Location filter objects
- Algorithm objects
- Compositor objects
- Output objects

These objects must be combined in order to create a complete viewshed calculation pipeline. Basic implementations of these objects are provided so that a calculation can be performed with minimal effort.

Except for the algorithm object an application may provide custom implementations of these objects in order to provide application-specific input data and display results in an efficient manner.

17.9.1 Input objects

Input objects (`TSLTerrainVSInput`) define the interface used by the viewshed algorithms in order to retrieve source data. The input object exposes terrain data as a 2-dimensional array of doubles and a geographical extent.

The following implementations are provided:

- `TSLTerrainVSInputArray` – This basic implementation can expose application-provided data to the viewshed algorithm.
- `TSLTerrainVSInputTerrainDatabase` – This is a more advanced input object, which exposes a MapLink terrain database (`TSLTerrainDatabase`) to the viewshed algorithm. In a similar manner to the terrain data queries, this object will select an appropriate level of detail from the database for the desired output size/extent.

- `TSLTerrainVSInputEarthCurvature` – This class wraps an existing input object and applies height corrections to compensate for earth curvature. Curvature corrections can be based on a visual or radar line of sight.

17.9.2 Location Filters

The location filter interface (`TSLTerrainVSLocationFilter`) is provided as a means of limiting the viewshed parameters to specific locations. The exact manner in which these filters are applied depends on the specific viewshed algorithm. Some algorithms may apply the filter to every input point, but some may not.

The API documentation for each viewshed algorithm specifies how these filters are applied to the process.

17.9.3 Algorithm Objects

Algorithm objects (`TSLTerrainVSAlgorithm`) contain the core viewshed functionality. They are responsible for performing calculations with the specified parameters and using the provided input/filter/compositor/output objects as needed.

The following viewshed algorithms have been provided:

- `TSLTerrainVSAlgorithmRFVS` – An algorithm object, based on the RFVS [Frankil and Ray 1994] algorithm.

The following parameters may be used when calculating a viewshed:

- Start/Centre point
- Start height (Absolute or relative to ground)
- End height (Absolute or relative to ground)
- Maximum Radius

The resolution and extent of a viewshed output is a combination of the input object's extent/resolution, and the specified parameters.

17.9.4 Compositor and Output Objects

Compositor objects (`TSLTerrainVSCompositor`) are provided with the results of the viewshed calculation. Each point in the calculation is passed to the compositor, along with:

- Whether the point can be seen from the viewshed's centre.
- The height of the point, including any height offsets/corrections applied during the calculation.

The following compositor object implementations are provided:

- `TSLTerrainVSCompositorVisibility` – A basic compositor which will store the visibility of each point in the output object.

- `TSLTerrainVSCompositorCumulative` – A basic compositor which will store the visibility of each point in the output object. This compositor can be used to accumulate the output of multiple viewshed calculations, in order to determine which areas can/cannot be seen from a set of points.

Output objects (`TSLTerrainVSOutput`) are used for storing result data, by the provided compositor objects.

The following output object implementations are provided:

- `TSLTerrainVSOutputArray` – A basic output object which will store data in a 2-dimensional array.

The provided implementations are designed so that any compositor can be used with any output object. This allows an application to perform viewshed calculations with minimal effort, however the generic interface between the compositor and output objects results in reduced calculation performance.

Most applications should implement their own compositor object. This allows the viewshed results to be rendered immediately without first going through an output object.

17.9.5 Single Point-to-point Line of Sight

```
// Set up the input object, using the current display size, and extent
// The provided extent/display size will determine which level of detail
// is used within the terrain database.
TSLTerrainVSInputTerrainDatabase* input =
  new TSLTerrainVSInputTerrainDatabase(&m_terrainDatabase,
                displayExtent, displayWidth, displayHeight);

// Setup the viewshed algorithm
// For single LOS calculations, the RFVS algorithm doesn't require
// a compositor object.
// It does however require an input object, to calculate whether
// the line is blocked by terrain.
TSLTerrainVSAlgorithmRFVS algorithm(input, NULL);

// Determine whether the end point can be seen from the start point.
// If it cannot, the 'blocked' point will be populated
double LOSStartX = -122.0;
double LOSStartY = 37.0;
double LOSStartZ = 100.0;

double LOSEndX = -121.5;
double LOSEndY = 37.35;
double LOSEndZ = 0.0; // Sea level

// Storage for the 'blocked' point
double LOSBlockedX = 0.0;
double LOSBlockedY = 0.0;
double LOSBlockedZ = 0.0;

if( algorithm.calculateLineOfSight(
        LOSStartX, LOSStartY, LOSStartZ,
        LOSEndX, LOSEndY, LOSEndZ,
        LOSBlockedX, LOSBlockedY, LOSBlockedZ)
    == TSLTerrainVSAlgorithmRFVS::LOSResultBlocked )
{
  // The end point is not visible from the start point.
  // The point where the line is blocked is stored in
  // LOSBlockedX, LOSBlockedY, LOSBlockedZ
}
// The end point is visible from the start point.

// Release our references
input->dec();
```

17.9.6 Area Viewshed Using Provided Classes

The following example performs a single viewshed calculation using application-provided terrain data.

Note: As this example uses the generic compositor/output objects it is not the fastest method of performing a viewshed, nor the recommended approach for most applications. This approach should mainly be used when performance is not critical and the results do not need to be displayed immediately.

```
// Create an input object array, at the specified size/extent.
// Data should be populated from the application-specific source.
// 100 x 100 sample grid, covering -30.0,-60.0 to 30.0,60.0.
// The provided coordinate system is EPSG:4326 (A valid coordinate system must
// be provided for viewshed calculations)
TSLTerrainVSInputArray* input = new TSLTerrainVSInputArray(
    100, 100,
    TSLMUExtent(-30.0, -60.0, 30.0, 60.0),
    coordSys4326);

// Create an output object to store viewshed results.
// This object should be created with the same extent/size as the input object.
// The output may be re-used for multiple viewsheds, however this basic
// implementation only provides basic validation of the coordinates.
TSLTerrainVSOutput::dataItem defaultVal;
defaultVal.type = TSLTerrainVSOutput::typeTSLTerrainVSVisibility;
defaultVal.data.v = TSLTerrainVSVisibility::TSLTerrainVSNoData;

TSLTerrainVSOutputArray* output = new TSLTerrainVSOutputArray(
    input->width(), input->height(),
    input->queryExtent(), defaultVal);

// Create a compositor object.
TSLTerrainVSCompositorVisibility* compositor =
    new TSLTerrainVSCompositorVisibility(output);

// Create the algorithm object.
TSLTerrainVSAlgorithmRFVS algorithm(input, compositor);

// Perform a viewshed calculation.
// The format of the data, and interpretation of results is defined by the
// compositor object.
// In this case, the visibility of each point will be stored in the output
// array.
algorithm.calculateViewshed(
    centerX, centerY,
    startHeightOffset, startRelativeToGround,
    maximumRadius, endHeightOffset, endRelativeToGround);

// Release our references
input->dec();
compositor->dec();
output->dec();
```

17.9.7 Performance Considerations

The overall performance of the viewshed algorithm mainly depends on the efficiency of the input/compositor objects, the extent of the viewshed calculation, and the resolution of the terrain data.

The RFVS viewshed algorithm, as used in the terrain viewer, is generally fast enough to be considered 'real-time' when performing calculations up to 10,000 x 10,000 samples (Running on an Intel dual-core processor @ 3.6GHz). For example, if the source terrain data has a post-distance of 20m the performance should be acceptable with a maximum radius of up to 100km. If the terrain data is of a higher resolution, or the maximum radius parameter is increased, then the performance will be decreased accordingly.

If using the `TSLTerrainVSInputTerrainDatabase` class an appropriate level of detail will be selected from the terrain database. However, the lowest detail in the database may still be too high if attempting to perform a very large viewshed. Where possible the post distance of the data should be checked and the viewshed radius limited accordingly.

The RFVS viewshed algorithm will use multiple threads where available so any application-defined objects must be threadsafe. The maximum number of threads used by the algorithm may be specified via the `TSLTerrainVSAlgorithm::maxThreads` method.

17.9.8 Application Integration

This section details how to integrate the viewshed functionality with an application. This is intended for applications which need to overlay viewshed results on a map display.

An example of this is provided by the 'Terrain Viewer (C++ and MFC)' sample application.

This application allows the following viewshed parameters to be adjusted:

Start/centre point

Start height (Absolute or relative to ground)

End height (Absolute or relative to ground)

- Maximum Radius

Correction for earth curvature (Visual or radar)

Result visualisation colours

17.9.9 Input Object Setup

If using a MapLink terrain database, the application simply needs to create an instance of `TSLTerrainVSInputTerrainDatabase`.

`TSLTerrainVSInputTerrainDatabase::displayParameters` should be called before performing each viewshed, to ensure that the correct extent/data resolution is exposed via the input object. In the terrain viewer, this happens as part of `TSLViewModeArea::performViewshed` and `CTerrainViewerDoc::calculateViewshed`.

The input object will query data from the nearest level of detail. This means that the viewshed calculation will be performed at the data resolution, and the results will need to be resized by the compositor when drawn to the screen.

The application may use other sources of data through the provided `TSLTerrainVSInputArray` class or by providing an application-specific input object.

The terrain viewer also creates an instance of `TSLTerrainVSInputEarthCurvature`. This is used as the input object if earth curvature corrections have been enabled in the options. Enabling earth curvature correction will decrease the viewshed performance as it adds an additional calculation to each sample point.

17.9.10 Application-Specific Compositor

In order to display the viewshed results the terrain viewer defines the `CTerrainClientCustomDataLayer` class. This class inherits from both `TSLTerrainVSCompositor`, and `TSLClientCustomDataLayer`.

When a viewshed calculation is performed the results are passed to `CTerrainClientCustomDataLayer::setData`. This method stores the viewshed results directly into a Windows HBITMAP. This is then drawn over the map via the `TSLClientCustomDataLayer::drawLayer` method.

This approach enables the application to perform calculations without making unnecessary copies of the data and with a minimum number of function calls. It also allows the rendering of the results to be controlled as a MapLink data layer.

The compositor in the terrain viewer has been designed for single-threaded use only. If an application is going to perform viewsheds over a large area they should be performed in a background thread in order to keep the application responsive.

18 MapLink 3D Earth SDK

The MapLink Pro Earth SDK provides a MapLink 3D API to customer applications, building upon the powerful and performant OsgEarth 3D library.

18.1 Sample Application

A sample application is provided to demonstrate capability and the usage of the API. The sample code may be found in the installation at Samples/NT/3DEarth.

18.1.1 Interaction Modes

The sample application demonstrates various methods of interacting with the scene, accessible through the sample application's Interaction Mode menu. When a menu entry is selected that mode will be activated and remain active until the user selects another.

The implementation of these interaction modes may be found in the interactions folder within the sample application.

18.1.2 Trackball View Interaction

A mode which provides the ability to navigate the globe using the mouse pointer.

When active (by the default settings), the following actions can be performed:

- Pan: Click and drag the left mouse button across the globe to move the camera's eye and target together across the map. Movement is relative to the actual movement of the mouse pointer across the map.
- Tilt: Click and drag the right mouse button to change the orientation of the camera's eye relative to the target. Vertical mouse movement raises or lowers the tilt angle, and horizontal movement changes the heading.
- Zoom: Scroll with the mouse wheel to decrease or increase the distance between the camera and the target. Forward scrolling brings the camera closer to the target.

An alternative constructor has been provided to allow remapping of Pan and Tilt operations to different mouse buttons.

The sensitivity (the factor by which mouse movement affects the camera movement) can also be changed for the Tilt and Zoom operations. This can be done in the alternative constructor, or via separate setter functions.

18.1.3 Select Geometry/Track

A mode which provides picking/selection functionality of tracks and geometry primitives. When active a left click will perform picking operations on the scene bound to the left mouse button.

If a Track is clicked it will become the active track and display additional information.

If a geometry instance is clicked its rendering will toggle the primitive's style between a normal and selected variant, which primarily affects the colour of the primitive.

18.1.4 Create Polygon

A mode which creates earth::geometry::Polygon primitives within the scene.

The left mouse button will start a new primitive or add points to the in-progress one. The right mouse button will finish creation of the primitive and add the final version into the scene.

These polygons will be created as draped 2D primitives and will automatically follow any underlying terrain.

18.1.5 Create Polyline

A mode which creates earth::geometry::Polyline primitives within the scene.

The left mouse button will start a new primitive or add points to the in-progress one. The right mouse button will finish creation of the primitive and add the final version into the scene.

These lines will be created as draped 2D primitives and will automatically follow any underlying terrain.

18.1.6 Create Text

A mode which creates earth::geometry::Text instances within the scene.

The left mouse button will create a new text instance at the clicked location, with the value 'Test Text'.

18.1.7 Create Symbol

A mode which creates earth::geometry::Symbol instances within the scene.

The left mouse button will create a new symbol instance at the clicked location, displayed as a MapLink vector symbol.

18.1.8 Create Extruded Polygon

A variant of the polygon mode which creates extruded primitives. These primitives are not draped over the terrain but are extruded to form 3D volumes.

The left mouse button will start a new primitive or add points to the in-progress one. The right mouse button will finish creation of the primitive and add the final version into the scene.

18.1.9 Create Extruded Polyline

A variant of the polyline mode which creates extruded primitives.

These primitives are not draped over the terrain but are extruded to form vertical walls.

The left mouse button will start a new primitive or add points to the in-progress one. The right mouse button will finish creation of the primitive and add the final version into the scene.

18.1.10 Delete Geometry

A mode which deletes geometry primitives from the scene.

Clicking the left mouse button on a geometry instance will remove it from the scene.

This interaction will have no effect on tracks.

18.2 API usage

18.2.1 Layer loading

To load a map file the user needs to do the following:

- Create a TSLMapDataLayer;
- Call loadData on the layer and provide the path to the .map file;
- Add the layer to the earth::Surface3D.

See CEarthSampleDoc::loadMapLayer and CEarthSampleView::addDataLayer in the EarthSample project for usage examples.

18.2.2 Terrain Loading

To load a Terrain database into the scene, the user needs to:

- Create a TSLTerrainDatabase;
- Call the open function on the database, providing a path to the database file;
- Add the database to the earth::Surface3D.

See CEarthSampleDoc::loadTerrainDatabase and EarthSampleView::addTerrainDatabase in the EarthSample project for usage examples.

18.2.3 Camera Movement

To update the camera, the user needs to do the following:

- Query the camera from the earth::Surface3D;
- The position of the camera and other such variables can be updated by calling the relevant functions on the class;
- To set the camera's target position, the easiest way is to use the lookAt function which sets the camera's eye and target position simultaneously, with roll angle as an additional option.

See CCameraControlFormView::updateCamera in the EarthSample project for usage examples.

18.2.4 Track Management

To simulate tracked objects on the globe, the user needs to:

- Create earth::TrackSymbol visualisations, which define how a Track will be rendered;
- Create earth::Track objects that use those visualisations, and store them for later use;
- Add each Track to the earth::Surface3D (which will use a pointer to the original Track objects);
- The Tracks can then be updated at run time using the variable and attribute functions on the Track objects.

See CEarthSampleDoc::initialiseTracks, CEarthSampleView::addTracks, and CEarthSampleDoc::updateTracks for usage examples.

18.2.5 Managing Geometry

To display geometry, the user must first configure a Style. This concept is similar to 'feature rendering' within the Core SDK but provides a distinctly different set of rendering parameters.

- A style is defined by an instance of the earth::geometry::Style class;
- Unlike the Core SDK a default style is provided. This is the initial state of the Style class, and will be used if a geometry references an unknown/invalid style on the surface;
- Once configured the user must pass the style to Surface3D::setStyle.

The set of styles may be configured independently on multiple surfaces. If a geometry instance is present in multiple surfaces it will be rendered according to the style on each one.

Once a style is configured geometry may be created by:

- Creating an instance of earth::Geometry, such as a Polygon or Polyline;
- Defining the primitive's coordinates;
- Setting the styleName parameter of the Geometry to the name of a style;
- Passing the Geometry instance to Surface3D::addGeometry.

Once added to the surface the geometry may be updated at any time, and the scene will be refreshed automatically to reflect this.

Note that very frequent updates to the primitive or to the styles may cause a performance loss, so these should be kept to a minimum. The CreatePolygonInteraction and CreatePolylineInteraction classes within the sample demonstrate a technique to reduce

this impact by performing most of the geometry creation with a much simpler style (e.g. one that has less detailed draping and tessellation options).

As with Tracks the application is responsible for memory management of Geometry instances.

19 EDITOR SDK

The Editor SDK provides an easy way for a MapLink Pro application to support the creation, management and update of vector overlays. It is used in many applications ranging from complex CAD-style editing for property Title Deeds to complex mission planning tactical graphics.

19.1 Library Usage and Configuration

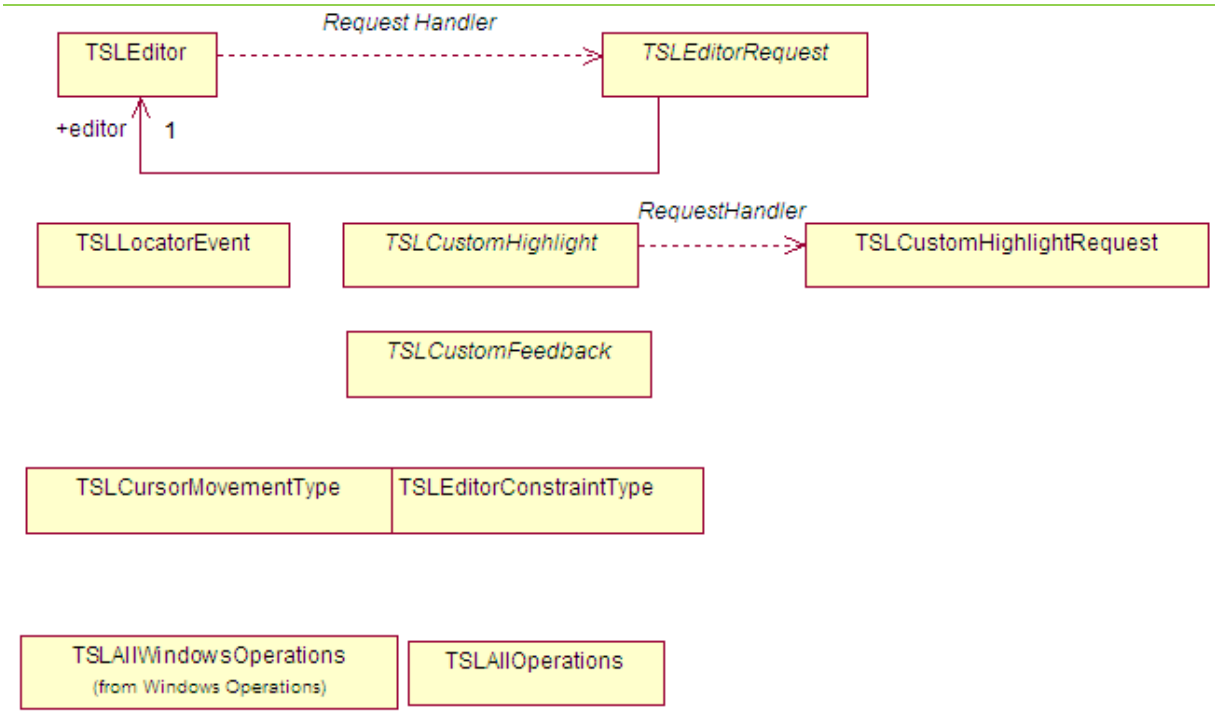
As of version 11.1, MapLink is no longer supplied with Debug or 32-bit libraries. Therefore, your application's build should link against the Release Mode libraries in all configurations.

MapLinkEDT64.lib Release mode, DLL version. Uses Multithreaded DLL C++ run-time library. Requires TTLDLL preprocessor directive. Refer to the document "MapLink Pro: Deployment of End User Applications" for a list of run-time dependencies when redistributing.

19.2 Concepts

The Editor SDK adds two new key concepts to those you will already be familiar with:

- Operations
- Select List
- The primary application interface goes through two principle classes:
- Editor manager – `TSLEditor` for instructions from the application to MapLink
- Editor request handler – `TSLEditorRequest` for requests from MapLink to the application.
- Custom Operations allow an application to evolve and expand the functionality of the SDK using two further main classes
- User operation – `TSLNUserOperation`
- Operation request handler – `TSLNUserOperationRequest`



19.2.1 Operation

In the Editor SDK, an operation encapsulates all functionality required to implement a user interaction. There are four main styles of operation:

1. One-shot style – e.g. "Delete all selected entities"
2. Creation style – e.g. "Allow me to draw a polygon". These operations will automatically clear the select list when triggered and are expected to add the created object to the select list.
3. Manipulation style – e.g. "Allow me move points of selected entities". These operations will automatically deactivate and current operations and are expected to manipulate the select list or the entities referenced by it.
4. Attribute style – e.g. "Set the fill colour of a selected polygons to red". These operations are expected to be transitory and perform simple, non-destructive changes to the entities on the select list. They leave any existing operation as active. If they are invoked with nothing on the select list, then most attribute operations will be expected to store 'default' values. They are often invoked by 'creation' options to apply current styling to the newly created objects.

They can response to events passed by the application and make requests for user input or application action where necessary. In general, the standard operations follow the "*Select objects, then select action to perform on the objects*" paradigm.

19.2.2 Select List

The Select List is an ordered list of entities that the user has clicked on. In addition to the actual fact of selection, the Select List also includes information about whether a specific vertex or point on the entity was selected.

It is managed by the Editor SDK, but used, updated, and manipulated by individual operations.

19.3 Editor Application Architecture

MapLink has unparalleled flexibility amongst GIS components. Virtually all MapLink Pro SDK classes are passive in that they:

- Do not create windows
- Do not force a main loop on the application
- Do not trap events
- Do not generate events

Exception: Some Editor SDK operations will redraw the window, unless the application has provided a mechanism to override that behaviour and redraw the window on the SDKs behalf.

This design increases portability and flexibility, but your application needs to pass on key events to the SDK, including initialisation, resize, paint and mouse events.

19.3.1 Limitations and Interaction with Other MapLink Pro SDKs

You can only have one `TSLEditor` instance per application, to which you must add the operations you want available. The `TSLEditor` can only be attached to one `TSLDrawingSurface` at a time, although it can swap to another.

The Editor SDK will only edit vector TMF Geometry stored in a `TSLStandardDataLayer` and will assume that it should create, select and modify data in the topmost editable, selectable `TSLStandardDataLayer`:

- `TSLPropertyDetect` is true on the Drawing Surface data layer properties.
- `TSLProperySelect` is true on the Drawing Surface data layer properties.

Before changing or storing contents outside of the editor (e.g. via load or save), you must call reset to clear select list, which will remove highlighting and any held references. After changing the contents of the editable layer outside of the editor, you must call `TSLEditor::dataChanged` to reattach the editor to the edit layer.

19.4 User Interface Considerations

When deciding to integrate the Editor SDK into your application, you will need make some key decisions about how your application will handle user interaction. For instance:

- Which operations are required or useful? Too many can lead to an unnecessarily complex user interface.
- Standard / custom operations. Are the standard variants appropriate, or do you need to provide additional functionality.

- Display of prompt messages and errors. How is this done? Typically, this might be through status bar messages, but this may not be appropriate.
- Handling of user entry – dialogs and text. Some operations may need a user to enter some text or make a choice from a number of options. How should this happen?
- Highlighting and selection of entities. What style of highlighting and selection is required? The Editor SDK provides two different standard ways – one CAD-style, the other more like a traditional Windows drawing application. Both can be adapted for rendering styles and colours without any code.
- Inter-relationship with view handling – zoom/pan/grab etc – This is especially important when using Context Menus. The standard Interaction Modes can be used for a simple integrated solution.
- Full feedback and GUI selection of entities. The standard operations allow a user to select entities by clicking on them. By using the feedback handlers, an application can maintain a separately displayed list of current objects and use that to interact with the select list.
- Configuration of rendering attributes. Does your application work with a limited set of rendering styles and colours for a particular domain, or does it need full user control over styling?
- Advanced point handling – point/object snapping, constraints etc

19.5 Configuration

In addition to the standard Core SDK styling configuration files, the Editor SDK requires two more files:

- A configuration file specifying setup information
- A messages file for text strings passed to the application

These can easily be customised or swapped at start up to support other locales.

19.5.1 Configuration File Format

Look at the example configuration file for the one of the Editor SDK samples. For example, `<MAPLINK_HOME>samples\NT\SimpleEditorInteraction\editor.ini`

The only section required by the `TSLEditor` is the `[editor]` section. This may be combined with other sections for an application. All the settings under the editor section have documentation to describe what they do.

Adjacent to the sample's configuration file is its messages file, `tms_msg.msg`. The first section lists the categories of the entries. Only two are used currently TMS_MSG_INFO and TMS_MSG_ERROR. The second section lists the entries in the format: Number, Identifier, Category and Message Text.

e.g. 220 AP_EXTRUSION TMS_MSG_INFO Select start of section to extrude

Use the 'msg2header' utility to generate a header file from this, which can then user used by the application to inform the Editor SDK which prompts and errors to display. This separation of messages and code allows an application to select from several localised message files if necessary.

19.6 Editor Management

The `TSLEditor` class provides the application with its main interface into the Editor SDK. It provides several types of method for the application to control the Editor and its operations.

Some methods are for configuration:

- `initialise`
- `addCustomHighlight`
- `enableGlobalUndo`

Some methods are event drivers:

- `activate`
- `locator`

Some methods are queries for user interface feedback:

- `activatePossible`
- `querySelectedAttributes`

The `TSLEditor` instance is attached to a `TSLDrawingSurface` using the '`attach`' method and edits objects in the top-most, selectable, detectable `TSLStandardDataLayer`.

19.7 User Interface Handling

The `TSLEditorRequest` class allows the application to control its own user interface. An application must provide a derivation of this class to the `TSLEditor`. All Editor SDK classes and operations use these handler functions to request that the application supply some information to, or request information from the user. This covers:

- Entry of some text
- Display of prompt messages and error strings
- Choose from a set of options
- Event feedback – redraw, cursor movement
- Cursor style changes

19.8 Activating Operations

Operations are activated by name, which is defined in the operation's class documentation. Please refer to the MapLink API documentation. Typically, the names is all lower case.

```
m_editor->activate("polygon");
```

An application can pass in an object as user data for an operation. The operation's class documentation will note where this is necessary and the type that is required. Any user data is stored by the current operation and may be queried – although some operations don't permit this.

```
TSLRenderingAttributes ra ; // Now configure rendering attributes
editor->activate("renderingattributes", &ra);
```

19.9 Integrating the Editor SDK from First Principles

Starting with a copy of the MFC Sample which only displays a map, we will now add an Editor SDK from scratch, using a separate dialog to display feedback and messages. In your own application, this could perhaps be in a status bar, as per the more complex Editor samples.

NOTE: The next section proves a simpler way to integrate the Editor SDK for an application that already use the standard TSLInteractionModes, such as an instance of the Simple Interaction sample, or an application created using the Visual Studio wizard.

19.9.1 Set up the application configuration

Add the Editor SDK libraries to the relevant project linker settings. Add an include to "stdafx.h" for "MapLinkEDT.h" (just after the existing "MapLink.h) which will pull in all of the relevant Editor SDK classes and types.

In the sample Doc class, instantiate a TSLStandardDataLayer once a map is loaded. Add/remove the data layer to the drawing surface when the map data layer is added/removed. Set the TSLPropertyDetect and TSLPropertySelect properties on the data layer to both be true.

19.9.2 Provide prompt capability for the Editor SDK

Create a small dialog to display the editor feedback messages for operation prompts and select list count. In your application main window, instantiate one of the feedback dialogs.

Create a new class and derive it from TSLEditorRequest. In its constructor, pass the instance of the feedback dialog and override the following methods:

- TSLEditorRequest::displayPrompt – Use the parameter to set the text for the prompt label in the feedback dialog.
- TSLEditorRequest::displayError - Show a message box containing the error message.
- TSLEditorRequest::onSelectionChanged - Set the text for the select list label.

19.9.3 Initialise the Editor

Construct an instance of the TSLEditor, passing your TSLEditorRquest derived class. Call initialise on the TSLEditor instance, passing the full path to the .ini file.

Call attach on the TSLEditor instance, passing the drawingsurface.

Call dataChanged on the TSLEditor instance, to attach to the edit layer.

Add all standard operation to the editor – TSLAllOperations.add(editor).

Call reset to on the TSLEditor to start the default operation.

Activate the rendering attributes operation, passing the name “renderingattributes” and a populated TSLRenderingAttributes instance to define the initial rendering attributes.

19.9.4 Capturing and processing user interactions

Create a new derivative of the TSLViewMode class for interacting with the Editor – TSLViewModeEditor. In the new view mode, override Button and Mouse methods and pass the events on to the TSLEditor::locator method.

Note - positions must be converted from DUs to TMCs using the drawingSurace DUToTMC method.

Note - You will need to change the signature of the basic onMouseMove method to provide the currently pressed mouse buttons, since some operations required those.

19.9.5 Invoking Operations

Add a toolbar button to activate the editor mode, alongside the ‘pan’, ‘zoom’ modes.

Add ‘polygon’ and ‘delete’ toolbar buttons and in event handlers for those, call activate on the TSLEditor instance, passing the appropriate operation name.

19.10 Integrating the Editor SDK from using Standard Interaction Modes

Note: The ‘Simple Editor Interaction’ sample provides an example of these completed steps, with the additional provision of a Context Menu.

Start with a copy of the Simple Interaction sample of the Visual Studio generated sample application. Do the same steps as described in Sections 19.9.1 but also add ‘MapLinkEDTIMode.h’ as a required header file.

19.10.1 Initialise the Editor

In the same class that has provided the ‘TSLInteractionModeRequest’ overrides for the standard Interaction Modes, also derive from TSLInteractionModeEditorRequest – typically the View. This provides a single point of interface.

Provide overrides for displayError, displayPrompt and onSelectionChanged as per Section 19.9.3.

Where the standard Interaction Modes are created and added to the mode manager, (typically the View::OnInitialUpdate), also instantiate an instance of TSLInteractionModeEdit, passing in the path to the initialisation file and provide the TSLInteractionModeEditorRequest derivative. Add the new mode to the manager as the default mode.

19.10.2 Invoking Operations

Add a toolbar button to activate the editor mode, alongside the 'pan', 'zoom' modes. Add 'polygon' and 'delete' toolbar buttons and in event handlers for those, call activate on the TSLEditor instance that can be retrieved from the edit interaction mode, passing the appropriate operation name:

```
m_editMode->editor()->activate( "polygon", 0 ) ;
```

19.11 Custom User Operations

The TSLUserOperation abstract class allows the application to:

- Create brand new operations;
- Override or extent existing operations.

Methods are triggered automatically by the Editor SDK in response to events or application queries.

19.11.1 Types of Custom User Operation

User operations can be one of several types:

- Simple – add additional functionality on to an existing operation. e.g. Add extra data to newly created primitives;
- Duplicate newly created primitive in external data store;
- Perform additional validation;
- Aliased – create variants of existing operations. Allows original to be kept too;
- Custom – brand new operations, unrelated to existing operations. Could be similar, but with different interactions.

19.11.2 Custom User Operation Event Handlers

User operations will be called by the TSLEditor in response to user or application triggered events. They typically return the ID of a message that should be displayed as a prompt – specifically when a state machine driving the interaction may have changed. Return 0 to leave the prompt unchanged.

Overridable methods include:

- **activate** – Called when operation is activated, passed the input data.

- **activatePossible** – Called to check whether it's possible to activate this operation. E.G. The delete operation can only be activated when there is something on the select list.
- **backup** – Called to step back one step in the interaction. E.G. Go back one point when drawing a polygon.
- **backupPossible** – Called to check whether it is possible to go back!
- **constraintChanged** – Called when vertical/horizontal/equal/unequal constraint changes – This can affect echo.
- **deactivate** – Called when the operation is deactivated, usually when another is made active.
- **dialogEntered** – Called when the user responds to a dialog entry request.
- **done** – Called when the user indicates completion - usually by a right mouse button press.
- **locator** – Called when mouse event is passed to the editor.
- **reactivate** – Called to re-activate the already active operation, possibly with new activation data or to reset it back to its initial state.
- **requestHandler** – Called upon initialisation to attach operation to handler.
- **resetUndoBuffer** – Called to indicate that the any undo buffer should be cleared as another operation has since been invoked and undo data no longer required.
- **textEntered** – Called when user responds to text entry request via handler.
- **undo** – Called when user asks to undo previous action.
- **undoPossible** – Called to see whether it's possible to undo last action.

19.11.3 Custom Operation Support

The TSLUserOperationRequest class provides custom operations with their main interface into the Editor SDK. A custom operation should prefer to use this mechanism to interact with the application, in order to ensure that the Editor SDK remains in a known state.

The methods on this class allow the operation to:

- Query and control the select list;
- Pass feedback to the application or the user;
- Request user input;
- Configure dynamic (rubber band) echo;
- Configure static (point/line) echo;
- Access Editor configuration information.

19.11.4 Custom Operation Echo Styles

Many custom operations require echo of some description. These would be set by an operation calling 'setDynamicEcho' and/or 'setFixedEcho' where relevant on the interaction. Note that most dynamic echo styles will automatically update position based on the mouse move and current constraints – an operation does not need to update the echo itself in response to the mouse move event.

There are several types available, each in different styles:

- Dynamic Echo;
- Rubber band style, often (partially) moving with the cursor;
- Segments;
- Rectangles;
- Scaling rectangles;
- Corners;
- Spatial calculations – ray, parallel;
- Constraints automatically applied where relevant;
- Primitive echo;
- Points or polylines.

The styles and rendering attributes used to draw these are defined in the .INI file. Note that echo styles are reset between operations.

19.12 Advanced Editor SDK Topics

The following topics will be covered in more detail in a future release but see the API documentation for more information.

Custom feedback handler - Everything you ever wanted to know about what the Editor SDK and operations are doing. Must provide feedback in custom operations.

- GUI integration - activatePossible, backupPossible and undoPossible;
- querySelectedAttributes;
- Feedback handler for selection trees;
- Custom rendering attribute panels;
- Windows highlighting, selection and movement modes;
- Custom highlighting – Total control over highlighting.

20 GEOPACKAGE SDK

The GeoPackage SDK allows a developer to read, analyse and display GeoPackage data files.

Please contact support@envitia.com for additional information on the implementation.

20.1 Library Usage and Configuration

As of version 11.1, MapLink is no longer supplied with Debug or 32-bit libraries. Therefore, your application's build should link against the Release Mode libraries in all configurations.

MapLinkGeoPackage64.lib
Release mode, DLL version.
Uses Multithreaded DLL C++ run-time library.
Requires TTLDLL preprocessor directive.
Refer to the document "MapLink Pro X.Y: Deployment of End User Applications" for a list of run-time dependencies when redistributing, where X.Y is the version of MapLink you are deploying.

21 OWSCONTEXT SDK

The OWSContext SDK allows a User to read, analyse and display OWSContext documents data files.

Please contact support@envitia.com for additional information on the implementation.

22 MapLink OGC Services SDK

The MapLink OGC Services library was introduced in MapLink 6.0 and superseded the previously used frontend API of the MapLink Web Map Service (WMS). It was introduced to allow additional OGC Service implementations to be created and used through the same interface.

The SDK offers interfaces in C++, .NET and JAVA for the construction, configuration and use of the MapLink OGC Services. Currently the following services are offered by MapLink:

- The MapLink Web Map Service (WMS)
- The MapLink Web Processing Service (WPS)

It is intended that future versions of MapLink will offer additional services

An OGC Services Implementation, such as the MapLink WMS, is loaded as a plug-in to the MapLink OGC Services SDK at runtime.

The "MapLink OGC Services Deployment Guide" provides the most comprehensive instructions on deploying and configuring all the MapLink OGC Services. This section is intended to cover programming using the SDKs provided.

22.1 Library Usage and Configuration

The table below describes the pre-processor directives and link options that should be set in the Project Properties for using the MapLink OGC Service C++ API. For X11 targets, refer to the product Release Notes.

MapLinkOGCServices64.lib
Release mode, DLL version.
Uses Multithreaded DLL C++ run-time library.
Requires TTLDLL preprocessor directive.
No redistributable run-time available.
KEYED: Deployment machines only.

22.2 The MapLink WMS

The MapLink Web Map Service (WMS) is used to serve up user defined map data in a standardised format for use by client software across a network. It conforms to the 'Open Geospatial Consortium' (OGC) WMS standard version 1.3.0 but is also backwards compatible with all prior ratified versions.

Envitia supplies two types of installation of this component; a developer and a deployment version. The developer installation includes the debug versions of the WMS libraries to allow users to create their own plug-ins to serve custom data. Whereas the deployment installation includes the release versions of the libraries for deploying a MapLink WMS using the pre-built or user created plug-ins.

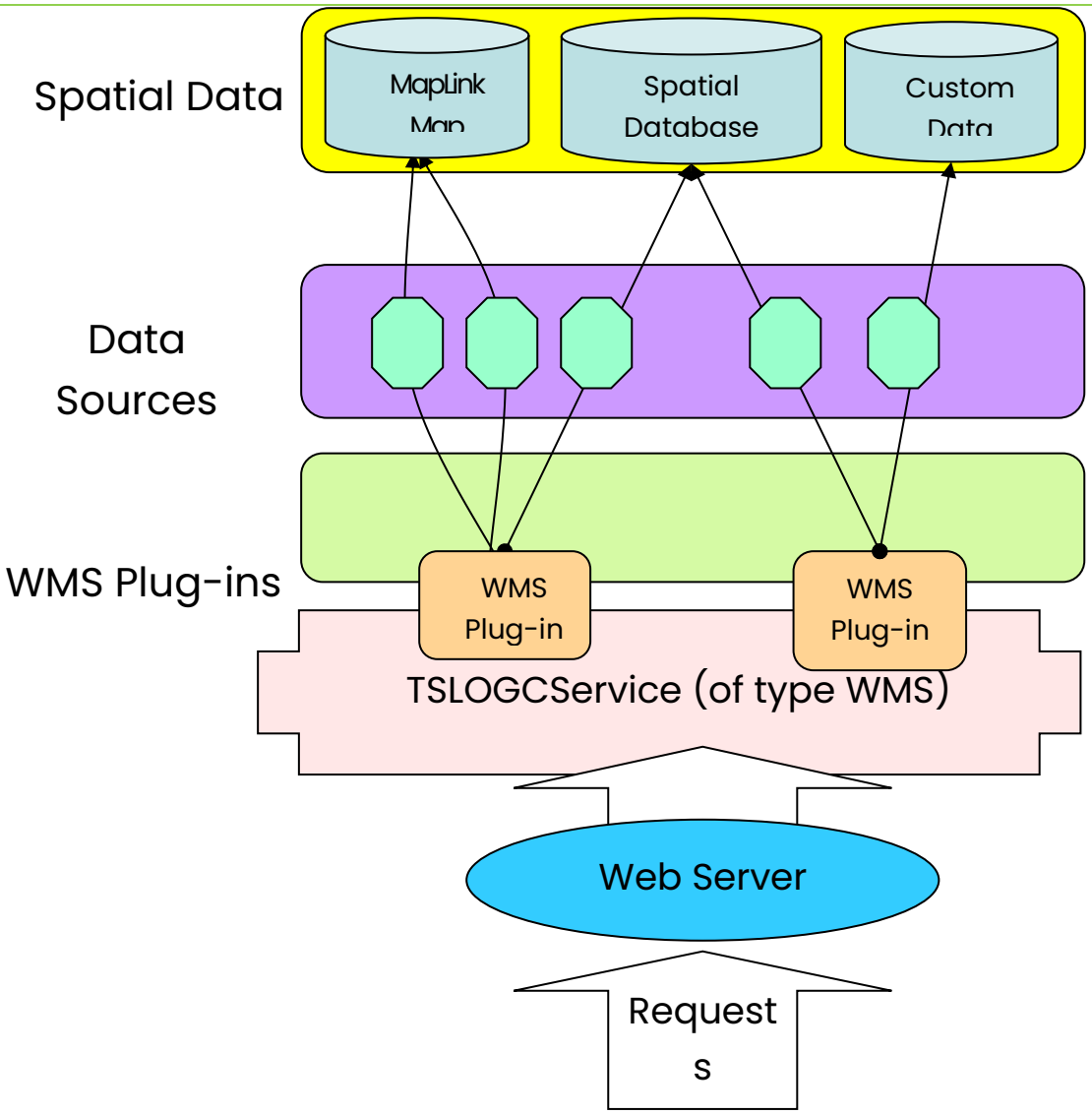
The "MapLink OGC Services Deployment Guide" provides instructions on how to deploy and configure your MapLink WMS on a variety of proprietary web servers. This section of the guide will cover the basic steps for creating your own plug-in to serve your own data.

22.2.1 Philosophy

As outlined in the OGC Services Deployment Guide, a WMS plug-in supplies the MapLink WMS with one or more data sources through its relationships to spatial data. A data source is the term hereafter used to refer to a child layer of the root layer in the WMS capabilities of the service. This is defined through a named combination of plug-in type and spatial data.

A single plug-in could potentially be used to serve the same spatial data in two different ways, thus creating two separate data sources. In practice, however, it is usually the case that a plug-in will be used to create separate data sources only when using different spatial data.

A good example of the use of a plug-in is the `BasicMapPlugin` supplied with the MapLink WMS. For spatial data it takes MapLink Maps, creating a different data source for each map.



The above diagram shows a set of example relationships between plug-ins, their spatial data and the data sources they provide.

22.2.2 Configuration

When the MapLink WMS starts up, it loads a single global configuration file that contains details of its root WMS capabilities as well as the data sources it is to load. The exact contents of this file are described in the OGC Services Deployment Guide, but this section will cover what details are passed to the plug-in.

Each data source is configured with three string entries in this global file; the plug-in name, the spatial data string and the data source configuration string. The MapLink WMS ignores the content of the latter two and only concerns itself with the former. It attempts to load the library of the plug-in name, appending '64' to the name when running in 64-bit mode and/or 'd' in debug, then queries the library for its `createDataSource` function.

If it fails to find this function, then the service will abort its loading and queries to the service will return a WMS exception report. If it's successful, it will pass the spatial data and data source configuration strings to this create function.

22.2.3 Library Usage and Configuration

Unlike most MapLink SDKs, when creating a custom WMS plug-in, the only library that must be linked against is the core WMS library. The table below describes the pre-processor directives and link options that should be set in the Project Properties for using the MapLink WMS SDK. For X11 targets, refer to the product Release Notes.

MapLinkWMS64.lib
Release mode, DLL version.
Uses Multithreaded DLL C++ run-time library.
Requires TTLDLL preprocessor directive.
No redistributable run-time available.
KEYED: Deployment machines only.

22.2.4 Plug-In Writing

As mentioned in the previous section, all WMS plug-ins must implement and export the `createDataSource` function for use by the MapLink WMS. An example of this is shown below.

```
#include "tslwmsplugindllspec.h"
#include "mydatasource.h"

extern "C"
{
  TSLWMSPluginDataSource* createDataSource( const char* spatialData,
                                            const char* dataSourceConfig )
  {
    try
    {
      return new MyDataSource(spatialData, dataSourceConfig);
    }
    catch (...)
    {
      return NULL;
    }
  }
}
```

The user-created plug-in should return an implementation of the abstract `TSLWMSPluginDataSource` class from the `createDataSource` function. The two abstract methods are the `getLayers` and `getMap` functions which must be implemented. Optionally the derived class may override the `getFeatureInfo` function if the plug-in is to support 'GetFeatureInfo' WMS requests.

The `getLayers` function is called immediately after the data source is created to build up the capabilities of the service. The data source should create and populate at least a root `TSLWMSAvailableLayer` object and potential sub layer objects. The MapLink WMS will internally handle how these objects are serialised to XML during the WMS 'GetCapabilities' requests. The `TSLWMSRegister` object passed to the `getLayers` function is for use when

overriding the `getFeatureInfo` function and is discussed later in this section. The `TSLWMSRequest` object passed to the `getLayers` function is for advanced usages and is discussed in the class documentation.

The `getMap` function is called whenever a WMS 'GetMap' request is made to the service. The data source implementation should examine the `TSLWMSGetMapRequest` object and populate the `TSLWMSGetMapResponse` object with its response. Currently the MapLink WMS only supports raster responses to 'GetMap' requests, but in future releases it is intended to additionally support vector responses.

For raster 'GetMap' requests, the response object should be cast up to the platform specific raster response object using the `isRasterResponseNT` and `isRasterResponseX11` as shown in the following example. The user can then either access the raster resource that this object represents or request a MapLink drawing surface based on the resource. Users should not create drawing surfaces independently due to thread safety issues discussed later in this section.

In the following example a pseudo implementation of these functions is provided.

```
TSLWMSAvailableLayer * MyDataSource::getLayers (TSLWMSRegister *wmsRegister,
                                                const TSLWMSRequest* r)
{
  if ( !m_isConfigurationValid  )
  {
    TSLWMSExceptionReport * report = new TSLWMSExceptionReport();
    TSLWMSCustomException * exception = new
        TSLWMSCustomException("MyDataSource not configured correctly");
    report->addException(exception);
    report->throwException();
  }

  TSLWMSAvailableLayer * rootMapLayer = new TSLWMSAvailableLayer();
  //TODO: Build up layer tree
  return rootMapLayer;
}

bool MyDataSource::getMap (TSLWMSGetMapResponse *response,
                           const TSLWMSGetMapRequest *request)
{
  if ( !m_isConfigurationValid  )
  {  //TODO: Throw exception report  }
#ifdef WINNT
    TSLWMSGetMapRasterResponseNT * res =
        TSLWMSGetMapRasterResponseNT::isRasterResponseNT(response);
#else
    TSLWMSGetMapRasterResponseX11 * res =
        TSLWMSGetMapRasterResponseX11::isRasterResponseX11(response);
#endif

  if ( !res )
  {
    //TODO: Throw exception report
    return false;
  }

#ifdef WINNT
  //TODO: Draw to response using either res->getHDC()
  //or res->getDrawingSurface()
#else
  //The implementation uses the DISPLAY environment variable for the
  //connection information.
  //Supported Visuals: True Color 24,16 and 8 bit depth, 8 bit Psuedo
  //Color.
  //TODO: Draw to response using res->getDrawable/Display/Screen/
  //Colourmap/Visual() or res->getDrawingSurface()
#endif
  return true;
}
```

22.2.5 'GetFeatureInfo' Usage

If any of the layers returned from the `getLayers` query to the data source have their 'Queryable' flag set to true, then the MapLink WMS expects the data source to provide an overridden implementation of the `getFeatureInfo` function. Failure to do so will lead to an exception report being generated if a service user performs a WMS 'GetFeatureInfo' request to that layer.

Unlike 'GetMap' requests, the MapLink WMS does not define any limitation on the type of response that a custom plug-in returns from a 'GetFeatureInfo' request. The only restriction it puts in place is that 'GetFeatureInfo' requests cannot query layers, in a single request, from multiple data sources. This is due to the MapLink WMS having no

understanding of the response and is therefore not able to merge multiple responses together as it does for 'GetMap' responses.

Plug-ins that wish to serve 'GetFeatureInfo' requests must register their supported output formats with the `TSLWMSRegister` object passed to the `getLayers` function at service start-up. Registered output formats will appear in the service level metadata returned from 'GetCapabilities' requests.

The `getFeatureInfo` method will be passed a `TSLWMSGetFeatureInfoResponse` object which it should populate with the raw data of the response.

22.3 The MapLink WPS

A Web Processing Service (WPS) provides a set of 'Processes', usually geospatial in nature, which take zero or more inputs and return one or more outputs. The WPS standard describes a process as 'any algorithm, calculation or model that operates on spatially referenced data', although its interface is not limited to geospatial operations. The WPS standard also provides an interface which will describe all processes WPS service.

Envitia's implementation of the WPS standard allows developers to build WPS plug-ins, each which can provide one or more WPS Processes. Each Process is defined as a class which inherits the 'MapLink WPS Data Source' class. The MapLink WPS Data Source class provides low level functionality which will interface with Envitia's WPS, allowing for the developer to concentrate on implementing the process itself.

Envitia supplies two types of installation of this component; a developer and a deployment version. The developer installation includes the debug versions of the WPS libraries, whereas the deployment installation includes the release versions of the libraries.

The 'MapLink OGC Services Deployment Guide' provides instructions on how to deploy and configure your MapLink WPS on a variety of web servers. This section of the guide will cover the basic steps for creating your own plug-in to serve your own data.

22.3.1 Library Usage and Project Configuration

When creating a custom WPS plug-in the table below describes the pre-processor directives and link options that should be set in the Project Properties for using the MapLink WPS SDK.

MapLinkWPS64.lib Release mode, DLL version. Uses Multithreaded DLL C++ run-time library. Requires TTLDLL preprocessor directive. No redistributable run-time available. Your application must also link: • MapLink64.lib • MapLinkwps64.lib • MapLinkows64.lib

22.3.2 Configuration

The WPS configuration file is used to define what WPS plugins exist. The contents of WPS configuration file is detailed in the OGC Services Deployment Guide. Each plugin is declared with three string entries:

- plug-in
- data path
- config path

The 'plugin' value defines the name of the DLL library to dynamically load.

The 'data path' and 'config path' are values that are passed to the plugin on start-up, allowing for customisation at a deployment level.

22.3.3 WPS Start Sequence

When the MapLink WPS starts up, it loads the WPS configuration file to determine which plugins to load. All WPS plugins must located in the 'plugins' sub folder of the appropriate bin folder.

Once found, the WPS service will attempt to find the DLL's `createAllDataSources`, or `createDataSource` function. If one of the functions is found in the DLL, the WPS will call this function and pass the 'data path' and 'config path' values found in the configuration file to it. If both functions are found, the `createAllDataSources` function will take precedence.

It is then the plugin's responsibility to provide class which inherits the `'TSLWPSPluginDataSource' class for each process the plugin is to provide.`

22.3.4 Plug-In Implementation

A WPS Plugin provides one or more `Plugin Data Sources` back to the WPS service when it starts up. Each Plugin Data Source being an implementation of the abstract `TSLWPSPluginDataSource` class. The WPS Plugin will achieve this by providing one of two functions; `createDataSource` or `createAllDataSources` function.

`The createDataSource` function can return a single Data Source.

```
#include "tslwpsplugindllspec.h"
#include "mydatasource.h"

extern "C"
{
  TSLWPSPluginDataSource* createDataSource( const char* dataLocation,
                                            const char* configurationLocation)
  {
    try
    {
      return new MyDataSource(dataLocation, configurationLocation);
    }
    catch (...)
    {
      return NULL;
    }
  }
}
```

The `createAllDataSources` function can pass back several Data Sources.

```
#include "tslwpsplugindllspec.h"
#include "mydatasourcea.h"
#include "mydatasourceb.h"
#include "mydatasourcec.h"
#include "mydatasourced.h"

extern "C"
{
  bool createAllDataSource( const char* dataLocation,
                            const char* configurationLocation,
                            TSLWPSDataSourceSet* dataSources )
  {
    try
    {
      dataSources->add( new MyDataSourceA(dataLocation, configurationLocation) );
      dataSources->add( new MyDataSourceB(dataLocation, configurationLocation) );
      dataSources->add( new MyDataSourceC(dataLocation, configurationLocation) );
      dataSources->add( new MyDataSourceD(dataLocation, configurationLocation) );
    }
    catch (...)
    {
      return false;
    }
    retrun true;
  }
}
```

22.3.5 Plugin Data Source Implementation

The two abstract methods are the `describeProcess` and `executeProcess` functions which must be implemented.

The `describeProcess` function is called immediately after the data source is created to build up the capabilities of the service and the process's description. The data source should create and populate a `TSLWPSProcessDescriptionType` object, describing what the process does, what inputs it takes and the type and format of the outputs that it can produce. The MapLink WPS will internally handle how these objects are serialised to XML during the WPS 'GetCapabilities' and 'DescribeProcess' requests.

The `describeProcess` function may be called multiple times if the server is configured to supported multiple languages from the service configuration file. It is important that the process description returned for each language is the same, except for the languages used to describe them, or the process may not operate correctly.

One of the settings on the `TSLWPSProcessDescriptionType` object is called '`storeSupported`' and is used denote whether the process supports asynchronous requests and referenced outputs. The MapLink WPS supports both and the plug-in may choose whether to permit their use, but another setting can affect whether the service advertised their offering; whether a data store has been configured in the service configuration. The data store configuration is detailed in the MapLink OGC Services Deployment Guide, which should be referred to for more information.

The `executeProcess` function is called whenever a WPS 'Execute' request is made to the service. The data source implementation should examine the `TSLWPSExecuteRequest` object and return a populated `TSLWPSExecuteResponse` object with its response.

If the process advertised that it supports reference outputs and the request object denotes that a supported output should be referenced, the `storeHelper` parameter will point to a valid `TSLWPSStoreHelper` class instance. This class's createStoreItem should be used to save an output and receive a URL through which the stored output can be retrieved. This URL should then be included in a `TSLWPSOutputReferenceType` instance, contained by a `TSLWPSOutputDataType` instance which should be added as a process output to the response.

If the process advertised that it supports asynchronous requests and the request object denotes that request of that type is being made, the plug-in need not concern itself with completing the request in a background thread. Instead the MapLink WPS will handle everything so that the plug-in need not handle the request any differently. The only exception is when the plug-in advertises that it supports status updates. In this case the `progressSink` parameter will be non-null and should be used to report the progress of the plug-in so that when a status request is made by the called, it can be responded to appropriately.

The following example code demonstrates the skeleton code for implementing a data source's plug-in.

```
TSLWPSProcessDescriptionType* SampleWPSDataSource:: describeProcess
                                                          (const char* language)
{
   if ( !m_isConfigurationValid  )
   {
     TSLOWSExceptionReport* er = new TSLOWSExceptionReport(1, 0, 0);
     TSLOWSException* ex = new TSLOWSException("NoApplicableCode",
                    "SampleWPSDataSource has not configured correctly");
     er->addException(*ex);
     ex->destroy();
     er->throwException();
   }

   TSLWPSProcessDescriptionType* desc = new TSLWPSProcessDescriptionType("1.0.0");
   desc->identifier().value("SampleProcess");

   //TODO: Add inputs and output descriptions

   return desc;
}

TSLWPSExecuteResponse* SampleWPSDataSource::executeProcess
                                          (const TSLWPSExecuteRequest *request,
                                           TSLWPSStoreHelper* storeHelper,
                                           TSLWPSProgressSink* progressSink)
{
  //TODO: Interrogate request for input values

  TSLWPSProcessDescriptionType * desc = describeProcess(request->language());
  TSLTimeType now;
  _time64(&now);
  TSLWPSStatusType* sts = new TSLWPSStatusType(now, "Succeeded");
  desc->identifier().value("SampleWPSDataSource");
  TSLWPSExecuteResponse * res = new TSLWPSExecuteResponse(*desc, *sts);
  desc->destroy();
  sts->destroy();

  //TODO: Add outputs

  return true;
}
```

23 SPATIAL SDK

The Spatial SDK introduces several additional spatial operations to the Editor SDK, such as the Arc, Parallel and Ray drawing modes and the Follow mode specialised operation. It also allows for the creation and merging of islands of change, which can be used to apply updates to a map.

23.1 Library Usage and Configuration

As of version 11.1, MapLink is no longer supplied with Debug or 32-bit libraries. Therefore, your application's build should link against the Release Mode libraries in all configurations.

LandLink64.lib Release mode, DLL version. Uses Multithreaded DLL C++ run-time library. Requires TTLDLL preprocessor directive. Must also link the MapLink CoreSDK library MapLink.lib Refer to the document "MapLink Pro X.Y: Deployment of End User Applications" for a list of run-time dependencies when redistributing, where X.Y is the version of MapLink you are deploying.

23.2 Islands

One of the principal uses of the Spatial SDK is to manage a series of updates to a vector based map. These updates can be grouped into a set of contiguous groups of entities called islands, each of which represents an independent area of change to the map.

23.2.1 Creating Islands

Islands are constructed via the two static `createIslands` methods available on the `TSLIsland` class. Both methods take a `TSLStandardDataLayer` containing the data to be converted into islands. Usually this layer will have been populated directly from a data source (such as a set of OS MasterMap COU files) via the interoperability functions as described in section 12.10. The act of creating the islands depopulates the source data layer, so if this layer may be required later a clone should be passed to the `createIslands` method instead of the original.

The first `createIslands` method performs a simple geometric merge of entities in the source data layer to create islands of contiguous features. This does not modify the geometry of any of the entities, it simply sorts them into contiguous regions.

The second `createIslands` method takes additional parameters that correspond to the full path to a map containing a seamless layer (see section **Error! Reference source not found.** for information on seamless layers), with an associated entity reference handler for the map. This method will replace 'departed' entities in the source data layer (entities which have a negative feature ID) with the geometry of the original entity from the map.

The departed status and the `TSLDataSet` of the entity from the source data layer will be preserved on the replaced entity. This may be useful in situations where having access to the geometry for departed entities is required.

By their nature, departed entities often overlap with the entities that replace them. However, they may not share any common edges and in this case the departed feature will be placed into a different island than the feature(s) that cover the same geographical location in order to prevent overlapping entities from being present in the island.

Should you wish these entities to be assigned to the same island that contains the features that replace it, you should call the static `sortDepartedFeatures` method on the `TSLIslandSet` populated by the call to `createIslands`. This method uses the geometry of the original entity in the map that is being deleted to reassign departed entities in the island set to the correct island.

23.2.2 Merging Islands

Multiple sets of updates to a map can be combined into a single update representing the sum of all the updates via the `mergeIslands` methods available on the `TSLIsland` class. There are two ways in which this can be done. The first performs a purely geometric merge of all entities contained within the source `TSLIslandSet`. This can be used to combine updates for different areas of a map into a single combined update for ingest into the map. This version of `mergeIslands` does not handle multiple different versions of the same entity being present in the source island set. If this happens, the resulting merged island set will still contain multiple versions of the entity.

This case is handled by the other two versions of `mergeIslands`. The difference between these two is that one allows merging of a new update contained within a `TSLStandardDataLayer` with the existing island set, while the other only merges the contents of the island set. The entities within the island set are merged so that only the newest version of the entity remains. In the case of merging a modified and departed entity the order in which the entities are encountered during the merge determines which remains in the merged island set. Thus, to ensure that the merge operation leaves the correct entity in the merged set, the following process should be followed:

1. Import the data corresponding to the first update into a `TSLStandardDataLayer`
2. Create an initial island set from the data using `createIslands`
3. Import the data corresponding to the second update into a `TSLStandardDataLayer`
4. Merge the initial island set and the data layer containing the second update using `mergeIslands`
5. Repeat steps 3-4 for each of the remaining updates.

The following code example illustrates this process.

```
// Load the map and initialise the seamless layer reference handler
TSLMapDataLayer *mapLayer = new TSLMapDataLayer();
mapLayer->loadData( pathToMap );
TSLSeamlessLayerConfig *config = new TSLSeamlessLayerConfig();
config->initialiseFromConfig( pathToConfig );
TSLSLMEntityRefHandlerFile *refHandler = new
TSLSLMEntityRefHandlerFile( *config );

TSLStandardDataLayer *initialUpdate = new TSLStandardDataLayer();

// Import the initial update data
TSLUtilityFunctions::importData( initialUpdate, … );

// Create the initial islands
TSLIslandSet *islandSet = new TSLIslandSet();
TSLIsland::createIslands(initialUpdate, *islandSet );

// Place departed entities into the same island as their replacement
// features
TSLIsland::sortDepartedFeatures( mapLayer, *refHandler, *islandSet );

TSLStandardDataLayer *couLayer = new TSLStandardDataLayer();

// Import the COU data
TSLUtilityFunctions::importData(couLayer, … );

// Merge the COU with the initial islands
TSLIslandMergeSet *mergedIslands = new TSLIslandMergeSet();
TSLIsland::mergeIslands( *islandSet, couLayer, pathToMap,
                         *refHandler, *mergedIslands );
```

23.3 Additional Editor Operations

These will be discussed in much greater detail in a future version of this document. Please contact Envitia support to see if there is a newer version available.

The intended topics are:

- Additional Spatial Operations
- Specialised Primitives
- Spatial Interactions
- Automatic Creation of Property Boundaries

24 GML SDK

24.1 Library Usage and Configuration

As of version 11.1, MapLink is no longer supplied with Debug or 32-bit libraries. Therefore, your application's build should link against the Release Mode libraries in all configurations.

MapLinkGML64.lib

Release mode, DLL version.

Uses Multithreaded DLL C++ run-time library.

Requires TTLDLL preprocessor directive.

Must also link the MapLink Core SDK library MapLink64.lib.

Refer to the document "MapLink Pro X.Y: Deployment of End User Applications" for a list of run-time dependencies when redistributing, where X.Y is the version of MapLink you are deploying.

The MapLink GML SDK is runtime locked meaning that before it may be used on any machine it must be unlocked programmatically. This is achieved using the `TSLUtilityFunctions` class by calling the `unlockSupport` method, passing the `TSLKeyedGML` enumeration value and the required unlock code. The unlock code can be provided on request from Envitia Sales, subject to licensing.

24.2 Supported Capabilities

The MapLink GML SDK offers the ability to read and write GML Application Schemas and corresponding instance data that either conforms to the 'Geography Markup Language (GML) Simple Features Profile' level 0 (SF-0) or GML of an equivalent complexity, that conforms to GML version 3.1.1.[9]

Non-SF-0 compliant instance data should use the '`FeatureCollection`' top level collection element.

If either the application schema or instance data is found to be incompatible during ingest, the GML library will attempt to continue but will disregard those feature definitions or feature instances that were not understood. The incompatible feature definitions or feature instances can be returned to the caller as self-contained blocks of XML if the relevant callback has been set.

A correctly defined GML feature definition should inherit from the `gml:AbstractFeatureType` type, which defines several properties that all derivates inherit. MapLink does not support the use of the base properties except for attribute `gml:id`. Instead the `gml:id` is added to all feature definitions and will be named '`gml:id`'.

[9] This section of the documentation, covering the use of the MapLink GML SDK, assumes a reasonable understanding of both GML and the GML SF-0 profile. For further information on these topics please refer to the OGC website (http://www.opengeospatial.org/)

MapLink requires that a GML instance data document only use a single coordinate reference system (CRS) for all geometry data. If a document uses more than one CRS, all uses of subsequent CRSs will cause the containing feature or features to be rejected.

24.3 GML Application Schemas

24.3.1 Schema Storage

The information read from a GML Application Schema, or used to write one, is represented by a `TSLGMLApplicationSchema` object. This class gives access to such properties as the version, target namespace, target namespace prefix and conforming GML profile of the schema.

The `TSLGMLApplicationSchema` class also offers the ability save the feature definitions it contains to a MapLink `TSLStandardDataLayer` object, through its `applyToLayer` method. The saving of feature definitions conforms to the following steps:

- Each feature definition is mapped to a MapLink 'feature' and can therefore be queried from the layer's feature class list. The name of the MapLink feature will be the same as the feature definition's enclosing XML tag.
- Each geometry property of a feature definition is added as a 'Source Info' property of the MapLink feature class. The details of these 'Source Info' objects can be queried via the `TSLFeatureClassList` class' `getSourceInfoItem` and `getSourceInfoCount` methods. 'Source Info' items can be added, removed or updated via the `TSLStandardDataLayer` class' `addSourceInfo`, `deleteSourceInfo` and `updateSourceInfo` methods.

The 'Source Info' details are mapped in the following way to each geometric feature property:

- The `sourceName` value corresponds to enclosing GML tags of the feature property
- The `sourceID` value corresponds to the zero-based index of the feature property. This allows the order of geometric and non-geometric feature properties to be maintained or determined.
- The `sourceDescription` value corresponds to GML annotation associated with the feature property, should one exist or wish to be set.
- The `sourceType` value corresponds to the type of GML geometry permitted. The following table lists the supported abstract GML geometry types and their corresponding values in the `TSLGeometryType` enumeration.

GML Type	TSLGeometryType value
Point	`TSLGeometryTypeSymbol`
Curve	`TSLGeometryTypePolyline`
Surface	`TSLGeometryTypePolygon`
Geometry	`TSLGeometryTypeEntity`

MultiPoint	TSLGeometryTypeMultiPoint
MultiCurve	TSLGeometryTypeMultiPolyline
MultiSurface	TSLGeometryTypeMultiPolygon
MultiGeometry	TSLGeometryTypeEntitySet

- The minOccurs and maxOccurs values correspond to the multiplicity of the property. Only positive values are valid, except for -1 for maxOccurs which is used to denote there being no upper limit.

  - Each non-geometry property of a feature definition is added to the TSLDataHandler of the Standard Data Layer, as a 'field definition'. The ways of interacting and the information stored in this class have been changed from previous versions of MapLink. MapLink 6.0 introduced a new class, TSLFieldDefinition, which represents a single 'field definition' or, in this case, a feature property definition.

A feature property definition maps to the fields of the TSLFieldDefinition class in the following ways:

- The name value corresponds to the enclosing GML tags of the feature property.
- The type value corresponds to the GML SF-0 supported type of which the field is defined as. The following table lists how the schema type supported by SF-0 is mapping to a TSLVariantType value

Schema Type	TSLVariantType value
xsd:integer	TSLVariantTypeLong
xsd:double	TSLVariantTypeDouble
xsd:string	TSLVariantTypeStr
xsd:date and xsd:dateTime[10]	TSLVariantTypeDateTime
xsd:boolean	TSLVariantTypeBool
Extensions of xsd:base64Binary and xsd:hexBinary	TSLVariantTypeBinary
xsd:anyURI	TSLVariantTypeURI
xsd:ReferenceType	TSLVariantTypeReference
Restrictions of gml:CodeType	TSLVariantTypeCode
Restrictions of gml:MeasureType	TSLVariantTypeMeasurement

- The minOccurs and maxOccurs values correspond to the multiplicity of the property. Only positive values are valid, except for -1 for maxOccurs which is used to denote there being no upper limit.
- The maxExclusive, maxInclusive, minExclusive and minInclusive values map to the schema facets of the same name. MapLink stores each value as a

[10] All xsd:dates and xsd:dateTimes are converted to UTC time

`TSLVariant`, which are meant to hold the appropriate value. The variant type should be the same as the field to which they belong.

- The `length`, `minLength` and `maxLength` values map to the schema facets of the same name.
- The `enumeration` value may hold an array of objects of the same type as the field to represent the enumeration schema facet.
- The `nillable` value is used to denote if the GML property is nullable.
- The `precision` value may hold the fraction digits schema facet value.
- The `encoding` field is used in certain cases to hold additional information. `gml:CodeType` types for instance store in this field their fixed or default constraint value, should one be defined.
- The `referenceTargetType` value is used to hold the `targetElement` value from a correctly formed `gml:ReferenceType` typed feature property declaration.

24.3.2 Schema Ingest

To load a GML Application Schema, first construct an instance of the `TSLGMLApplicationSchemaLoader` class. The following can optionally be set on the loader class before loading a schema:

- When loading a GML SF-0 compliant schema, it is possible to check that the schema is correct through using the `gmlSFValidationLevel` option. When turned on, this option will check the schema against the validation rules defined in the GML Simple Features profile.
- The `strictValidation` option can be used to perform more rigorous checks of the correctness of the schema. This is independent of any GML or GML SF-0 checks.
- The `unhandledFeatureDefinitionCallback` methods can be used to provide the loader with a callback that will be called whenever a feature definition is encountered that is not supported.
- During schema loading, any dependant schemas are also loaded using the URL referenced in the schema file. For cases when those addresses are no longer correct or inaccessible, the `urlLoaderCallback` methods allow a callback to be set that will redirect the loading address.

Finally the schema document may either the loaded from a file, URL or buffer using one of the `loadSchema` methods. On successful loading of a schema document, an instance of the `TSLGMLApplicationSchema` class will be returned.

24.3.3 Schema Creation and Export

To create a GML schema using the MapLink GML library, a `TSLGMLApplicationSchemaFactory` is used. It should be provided with instances of the `TSLStandardDataLayer` and `TSLGMLApplicationSchemaCreationParameters` classes.

The standard data layer should define the features that the schema should contain while the parameters class will contain the namespace, namespace prefix, version and GML profile to conform to. How to define GML features is described in section 24.3.1.

Optionally, a third parameter may be passed to the schema factory: a `TSLGMLPropertyMapping` or `TSLGMLPropertyMappingSet` class instance. These can be used to define additional feature properties whose values are determined by the rendering attributes of MapLink geometry. This concept will be described in more detail in section 24.4.1, but it is necessary to define those feature properties on the schema for instance data to conform to its application schema.

On successful creation of a schema, an instance of the `TSLGMLApplicationSchema` class is returned. Instances of this class can be used to write the schema to a file or buffer using the `TSLGMLApplicationSchemaWriter` class. All schemas written by MapLink will reference the `opengis.net` website for their dependent GML schemas.

24.4 GML Instance Data Ingest and Export

GML instance data can be loaded using MapLink via the `TSLGMLInstanceDataLoader` class. The information read from a GML instance data document, or used to write one, is stored in a MapLink `TSLStandardDataLayer`. The root `TSLEntitySet` will contain a child entity that represents each GML feature instance. This means that every GML feature instance read or written using MapLink must contain at least one geometric property.

As noted in section 24.2, all GML instance data either ingested or exported must be in a single coordinate reference system. When ingesting data, the `TSLGMLInstanceDataLoader` class will also return a `TSLCoordinateSystem` that represents that CRS.

Exactly how the information is store depends upon a number of factors, but one of the key ones is whether MapLink reads the instance data's schema. This is due to MapLink being capable of reading instance data without ever reading the schema; helpful if the schema is unavailable or not compatible with MapLink. When reading without a schema, MapLink treats all non-geometric properties as string values as it is not capable of determining their type. The following sections therefore deal with how the instance data is read and stored depending upon whether a schema was used to load it.

24.4.1 Schema Based Instance Data Ingest and Storage

The following rules define how the feature instance is stored as a MapLink entity:

- If the feature is defined as having more than one geometric property or the multiplicity of the geometric property allows for zero or multiple instances of the property, then the root MapLink entity that represents the feature instance will be a `TSLEntitySet`. Each child entity within the root entity set will have its new `dataSourceID` field set to denote the index of the 'Source Info' item to which it belongs.

For instance, if a feature defines two geometric properties; property 'A' with a multiplicity of 0..n and property 'B' with a multiplicity of 1..1. The instance data document contains just two feature instances. The first instance of that feature does not contain any 'A's but contains the required 'B' property. The second instance of that feature contains two 'A's and a 'B'. Therefore the standard data layer's entity set will contain two child entity sets; the first representing the first feature instance and the second set representing the second instance. The first child entity set, representing the first feature, will in turn contain one child entity representing property 'B' and that entity will have its `dataSourceID` set to 2. The second child entity set will in turn contain three child entities representing the two 'A' properties and one representing 'B'. These child entities will have the following `dataSourceID` values: 1, 1 and 2.
Alternatively, if the feature only defines a single geometric property with a `minOccurs` and `maxOccurs` of 1, then the geometry for that property will be used to represent the feature.

The reason for these two different schemes is to reduce the memory footprint of an application loading large amounts of instance data in the more common, latter, scenario.

- The `TSLDataSet` of the root entity representing the feature instance will be populated with non-geometric properties' values. By examining the standard data layer's `TSLDataHandler`, the two-character lookup key can be determined to discover the value of a particular property's value.

If a feature instance contains multiple instances of a property, then the dataset will be populated with multiple values for that property.
If the property is nil, then the `TSLVariant`'s `isNil` property will be set to true. Should a `nilReason` also be present, the variant will also contain a string value containing the content of this attribute.

- Using either a `TSLGMLPropertyMapping` or `TSLGMLPropertyMappingSet` class instance, it is however possible to map feature property values to the rendering attributes and other properties of the root entity representing the feature instance. Rather than a property's value being added to the `TSLDataSet`, it could for instance be set as the root entity's name or entity id.

The `TSLGMLPropertyMapping` class is used to setup a mapping for all types of features in the same way. Whereas the `TSLGMLPropertyMappingSet` class can setup a different mapping for each feature type.
The mapping object is not only used during reading of instance data but also when writing it. The process is simple reversed with the values of feature property's being determined from the root entity representing the feature.

MapLink offers two ways in which instance data can be loaded with a schema; Pre-load the schema using a `TSLGMLApplicationSchemaLoader` class or by loading the schema at the same time from the location referenced in the instance data document. The benefit of the former being that multiple instance data documents, that use the same schema, can be loaded serially far quicker than via the latter.

24.4.2 Schema-less Instance Data Ingest and Storage

The following rules define how the feature instance is stored as a MapLink entity:

- In the same way as the schema-based storage, the root entity used to represent the feature instance is determined by how many geometric properties the feature 'appears' to contain. The problem being that MapLink can only use the feature instances that the instance data document contains to make this determination. This can lead to two instance data documents, based upon the same original schema, being loaded and stored differently.

As with the schema-based loading, if all seen instances of a feature always contain a single geometry property, then that geometry object is used to represent

the feature in the MapLink standard data layer's root entity set. Otherwise a `TSLEntitySet` is used and all the geometric properties are added as child entities.

- The `TSLDataSet` of the root entity representing the feature instance will again be populated with non-geometric properties' values. All of the entries will be of type `TSLVariantTypeStr` type with the exception of correctly formed restrictions of '`gml:MeasureType`' which will be formed into `TSLVariantTypeMeasurement` types.

As all non-geometric properties are treated as strings, this means that only the content of the property will be stored. Any XML attributes on the enclosing property tags encountered when reading the instance data will be ignored.

- As with the schema-based loading, either a `TSLGMLPropertyMapping` or `TSLGMLPropertyMappingSet` class instance can optionally be used when loading without a schema. These mapping classes allow feature property values to be used to populate rendering attributes and other properties of the root MapLink entity representing the feature property.

24.4.3 Instance Data Ingest Options

The following options can be set on the `TSLGMLInstanceDataLoader` class to control aspects of the ingest:

- The `mapUnitShiftX`, `mapUnitShiftY` and `tmcPerMU` options provide control over how GML geometries are converted into TMC space.
- The `swapXandY` option sets whether the GML coordinates ingested should be treated as Y, X rather than X, Y.
- The `propertyMapping` and `propertyMappingSet` methods allow the settings of a mapping that will be used by subsequent loads. How these mappings alter the ingest process is described in section 24.4.1.
- The `unhandledFeatureCallback` method allows a callback to be set on the loaded that will be called whenever the loader encounters a feature instance that it cannot process. This may occur if the instance is of a feature type that is not supported by MapLink, contains a GML geometry type that is not supported or when a feature instance is encountered that is too complex during schema-less loading.

24.4.4 Instance Data Export

Exporting GML instance data requires it to be contained by a `TSLStandardDataLayer` in the same form as schema-based reading populates a layer. Exporting is performed by the `TSLGMLInstanceDataWriter` class and must be provided with both the `TSLStandardDataLayer` and a `TSLCoordinateSystem` class instance. The `TSLCoordinateSystem` describes both the CRS that the GML instance data will be written in and how to convert the TMC based geometry into Map Units (MUs).

MapLink also requires the schema to be provided to the exporter so that it can determine the format of the output. This schema can either be loaded using the `TSLGMLApplicationSchemaLoader` or created using the `TSLGMLApplicationSchemaFactory`.

Lastly, the export call can optionally be provided with a location at which the schema should be referenced. This location will be used to populate the `schemaLocation` XML attribute of the GML instance data document.

The following options are also available when exporting instance data and should be set on the exporter prior to the export call being made:

- The `swapXandY` option sets whether the GML coordinates exported should be treated as Y, X rather than X, Y.
- The `propertyMapping` and `propertyMappingSet` methods allow the settings of a mapping that will be used by subsequent exports. These mappings work in the reverse of how the ingest process uses them, thus the entity's properties are used to create the exported feature's properties. How these mappings alter the ingest process is described in section 24.4.1.

25 .NET SDKs

Developers familiar with the MapLink C++ API will find that the .NET APIs are very similar with the only major difference being the names of the classes.

The C++ API uses the class name prefix `TSL` to denote Envitia classes whereas the .NET libraries use the prefix `TSLN`. Other differences include the removal of Envitia helper classes such as `TSLSimpleString`, `TSLifstream` and `TSLofstream` which have been replaced with similar helpers from the .NET framework.

Furthermore certain 'getter' and 'setter' methods have been replaced by .NET properties or indexers.

Lastly the concept of colour indexes taken from the colour table has been hidden in the .NET wrappers meaning that when getting or setting colours the .NET framework class `Color` should be used.

Developers new to MapLink may wish to read the sections that deal with the basic use of MapLink in this document as the use of the .NET wrappers is almost identical. Although this section of the document will repeat some of these basic topics from a .NET view point, it won't cover them in such depth and is intended for users familiar with MapLink to assist getting to grips with its use via .NET.

25.1 Library Usage and Configuration

Currently MapLink supports .NET wrappers for the following SDKs with the library name and namespace listed:

SDK	Library Name	Namespace
Core SDK	Envitia.MapLink64.dll Envitia.MapLinkEx64.dll Envitia.MapLink.NativeHelpers.dll	Envitia.MapLink
OpenGL Drawing Surface SDK	Envitia.MapLink.OpenGLSurface64.dll	Envitia.MapLink.OpenGLSurface
OpenGL Track Helper SDK	Envitia.MapLink.OpenGLTrackHelper64.dll	
Direct Import SDK	Envitia.MapLink.DirectImport64.dll ttldirectimport.net64.dll	Envitia.MapLink.DirectImport
Interaction Modes SDK	Envitia.MapLink.InteractionModes64.dll	Envitia.MapLink.InteractionModes
Dynamic Data Object SDK	Envitia.MapLink.DDO64.dll	Envitia.MapLink.DDO
Editor SDK	Envitia.MapLink.Editor64.dll	Envitia.MapLink.Editor
Spatial Editor SDK	Envitia.MapLink.Spatial64.dll	Envitia.MapLink.Spatial
Terrain SDK	Envitia.MapLink.Terrain64.dll	Envitia.Maplink.Terrain

GeoPackage SDK	Envitia.MapLink.GeoPackage64.dll ttlgeopackage.net64.dll	Envitia.MapLink.GeoPackage
3D SDK	Envitia.MapLink.ML3D64.dll	Envitia.MapLink.ML3D
3D Interaction Modes SDK	Envitia.MapLink.InteractionModes.ML3D64.dll	Envitia.MapLink.InteractionModes.ML3D
OGC Services SDK	Envitia.MapLink.OGCServices64.dll	Envitia.MapLink.OGCServices
Rendering Attribute Panel	Envitia.MapLink.RenderingAttributePanel64.dll ttlrenderingattributepanel.net64.dll	Envitia.MapLink.RenderingAttributePanel
S52/S63 SDKs	Envitia.MapLink.S5264.dll Envitia.MapLink.S6364.dll	Envitia.MapLink.S52 Envitia.MapLink.S63

All other libraries are dependent on the Core SDK wrappers, Envitia.MapLink, while the Spatial Editor wrappers are also dependent on the Editor wrappers.

This is discussed further in the deployment guide, but it should be noted that these libraries are all wrappers around the C++ versions and therefore require the C++ libraries at runtime. This also means that the .NET wrappers will not work with the current versions of Mono, the .NET port to non-Windows operating systems.

The .NET version of the Spatial Editor SDK is the only SDK that does not offer all the functionality of its C++ counterpart. Several helper classes dealing with Islands have been omitted from this release. Later versions of MapLink may add this additional functionality.

25.2 C# Walkthrough 1 - Your First C# MapLink Application

By the end, you should have an application that can load and display a map generated from MapLink Studio and can correctly handle paint and resize events.

25.2.1 Skeleton Application

The starting point for this is a C# 'Windows Forms Application' Application Wizard generated executable. These instructions assume that the .NET Framework 4 is targeted.

Use the standard C# 'Windows Forms Application' Application Wizard to generate your skeleton application. The example application here is called "Hello Globe".

25.2.2 Configure Project Properties

Once created, build your skeleton application to ensure it compiles and links. You then need to import the appropriate MapLink .NET libraries into the project references. The MapLink installer does not integrate these .NET assemblies into the Visual Studio standard list of assemblies so you will need to browse to your installations bin directory, E.G.

```
C:\Program Files\Envitia\MapLink Pro\X.Y\bin64
```

Where X.Y is the version of MapLink you are using.

For the purposes of this walk through we will only import the Core MapLink .NET assembly `Envitia.MapLink64.dll`

NOTE: If you look at any of the projects for the C# samples that ship with MapLink you'll find that the MapLink .NET assemblies don't appear under the "References" node of the Solution Explorer. This is because the Visual Studio GUI doesn't support the x86 build configurations using one assembly and the x64 configuration using another. The underlying build system that Visual Studio uses, MSBuild, doesn't have such as limitation so they're included in the project but just don't appear in the GUI.

25.2.3 Initialisation and Clean Up

The first thing you'll need to do is add namespace declarations to the project's main form for all the newly added MapLink assemblies. This will mean that subsequent references to MapLink classes won't need to be prefixed with the namespaces that contain them. The namespaces for each of the MapLink assemblies are listed in section 25.1, e.g.

```
using Envitia.MapLink;
```

The configuration files for MapLink are usually only loaded once per execution run using static methods of `TSLNDrawingSurface`. In a C# application this can be done in a number of places, but the easiest is in the main form's constructor. The simplest way to go about this is to tell MapLink to load all standard configuration files from a particular directory. If no directory is specified, then MapLink will assume that a full MapLink installation has taken place and will attempt to load from there.

In the method constructor of the applications main form add a call to `TSLNDrawingSurface::loadStandardConfig`. This should be done before the call to `InitializeComponent` in case MapLink classes are constructed during this call.

You should be careful to check for, and report errors at this stage by using the methods supplied on the `TSLNErrorStack` utility class.

```
public Form1()
{
  TSLNErrorStack.clear() ;
  String configDirPath = null; //Replace if deployed
  TSLNDrawingSurface.loadStandardConfig(configDirPath);
  String msg =
    TSLNErrorStack.errorString("Initialisation errors :\n",
       TSLNErrorCategory.TSLNErrorCategoryError |
       TSLNErrorCategory.TSLNErrorCategoryFatal) ;
  if ( msg != null )
  {
    MessageBox.Show(this, msg) ;
    Environment.Exit(-1);
  }
  InitializeComponent();
}
```

When your application is deployed, make `configDirPath` variable point to the location of

Once MapLink has been initialised, it needs to be cleaned up when the application exits, otherwise Visual Studio will report numerous "leaks" which are in fact memory currently in

use when the application exits. This should be done in the `Dispose` method of the main form. This `Dispose` method is usually provided for you in the form designer but can be added via the class wizard if missing.

The tidy up of MapLink should occur after the form's components have been disposed of but before the form itself is disposed of.

```
protected override void Dispose(bool disposing)
{
  if (disposing && (components != null))
  {
    components.Dispose();
  }
  TSLNDrawingSurface.cleanup();
  base.Dispose(disposing);
}
```

25.2.4 The Drawing Surface and Map Data Layer

First of all we'll add some UI features to the main form to allow users to load a map:

- Add a Menu Strip to the form (Drag from the toolbox to the top of the form in designer mode)
- Add a 'File' menu group.
- Add an 'Open' and 'Exit' menu items to the File menu group. Add an event handler for each of the operations. (Click on the buttons to achieve this)
- Add an `OpenFileDialog` object to the main form. (Drag the icon from the toolbox to anywhere on the form). Set the filter in the properties window to:

```
MapLink Maps|*.map;*.mpc|All files|*.*
```

Also set the title and any other settings if required.

Next we'll declare the variables required and setup the drawing surface:

- Before the constructor in the main form's main class, declare private instances of both the TSLNDrawingSurface and TSLNMapDataLayer class and initialise them to null.
- Add an event handler for the main form's 'Load' event. This can be achieved via the properties window when the form is viewed from the designer by clicking on the events (the lightning icon) button at the top. When viewing the form events, find the Load event and type a method name next to it, or double click for the default method name to be used.
- In the load event handler construct the TSLNDrawingSurface, passing in the form's 'Handle' member variable and the second argument as false to indicate that the handle is a window handle. For example:

```
m_drawingSurface = new TSLNDrawingSurface(this.Handle, false);
```

Finally we'll hook up the menu event handlers to allow the map to load and the program to exit.

- In the File Open event handler call your instance of the FileOpenDialog's ShowDialog method. Capture the return value and return from the method if it's anything but DialogResult.OK.
- Construct your instance of the TSLNMapDataLayer class and call its loadData method using the filename retrieved from the user.
- Check the return value of the loadData call and if it fails check the error stack for the reason.
- Finally add the new data load to your instance of TSLNDrawingSurface via the addDataLayer method, assigning it a unique name.

For example:

```
private void openToolStripMenuItem_Click(object sender, EventArgs e)
{
  if (openFileDialog1.ShowDialog() != DialogResult.OK)
    return;
  m_mapDataLayer = new TSLNMapDataLayer();
  if (!m_mapDataLayer.loadData(openFileDialog1.FileName))
  {
    String error = TSLNErrorStack.errorString("Errors:",
    TSLNErrorCategory.TSLNErrorCategoryAll);
    m_mapDataLayer = null;
    if (error != null)
    {
      MessageBox.Show(this, "Error", error, MessageBoxButtons.OK);
      return;
    }
  }
  m_drawingSurface.addDataLayer(m_mapDataLayer, "map");
  m_drawingSurface.reset(true);
}
```

- Lastly when the File Exit menu button is clicked call the form's Close method.

At this point it would be advisable to compile the project to check if any coding errors have occurred so far. Although the program should run, you'll find that it won't do very much as we haven't told MapLink when it needs to draw yet!

25.2.5 Handling Paint and Resize Events

Since MapLink is passive, the application needs to handle relevant events and pass the information onto MapLink. Most applications will only need to handle the window paint and resize events.

A paint event can be triggered for many reasons, some of which will only want to redraw part of the window. Under these circumstances, Windows will set up a Clip Box to define the part that needs redrawing. To improve performance, it is best to only redraw that part. It is most efficient to pass the required Device Unit extent to the Drawing Surface.

After handling a resize event, Windows will usually post a paint message so there is no need to force a redraw in the resize handler. Just changing the window size may distort

the aspect ratio of the display, so MapLink can automatically adjust the visible map area to be in sympathy with the aspect ratio of the window. This optional behaviour allows an anchor point to be specified, which is kept at the same place when updating the visible map area.

To add `Paint` and `Resize` event handlers to your form, the steps are the same as outlined for adding a `Load` event handler detailed earlier in this walkthrough.

In the Paint event handler, first check that the drawing surface has been constructed and if so request a redraw via the `drawDU` method, e.g.

```
private void OnPaint(object sender, PaintEventArgs e)
{
  if ( m_drawingSurface == null )
    return;

  m_drawingSurface.drawDU(e.ClipRectangle.Left, e.ClipRectangle.Top,
                          e.ClipRectangle.Right,
                          e.ClipRectangle.Bottom,true);
}
```

In the `Resize` event hander, once again first check that the drawing surface has been constructed and if so, request a resize via the `wndResize` method. The second to last argument is whether a redraw should occur, which is required as .NET will only redraw if the control gets larger. The final argument to the `wndResize` method dictates how the existing view should relate to the new view, e.g.

```
private void OnResize(object sender, EventArgs e)
{
  if (m_drawingSurface == null)
    return;

  m_drawingSurface.wndResize(ClientRectangle.Left,
       ClientRectangle.Top,
       ClientRectangle.Right,
       ClientRectangle.Bottom,
       true,
       TSLNResizeActionEnum.TSLNResizeActionMaintainCentre);
}
```

Finally, we need to handle the initial resize of the main window. For some reason a Resize event is not sent by the .NET framework when the form is initially sized. So we'll have to tell the drawing surface its initial size. Simple copy the `wndResize` statement into your `Load` event handler after the drawing surface has been constructed.

Now build the program, run it and load one of the sample maps.

25.2.6 Further Tweaks

One of the problems with the setup used in the walkthrough is that the menu strip at the top of the application actually hides some of the map area. You'll notice this affect if you load a map as the area of white space at the bottom won't match the amount at the top.

In the sample C# applications supplied with MapLink we get around this issue by moving the drawing surface into a panel within the main form's client area. The same effect could, however, be achieved by sizing the drawing surface using the menu strip's bottom coordinates.

To add double buffering to the form in C#, users will need to override the form/panel's `OnPaintBackground` and not call the base implementation. This will be in addition to calling `setOption` on the drawing surface as described in section 8.10, e.g.

```
protected override void OnPaintBackground( PaintEventArgs pevent)
{
  // do nothing...
  // we don't want the background to flash over the map
}
```

NOTE: Override the `OnPaintBackground` method can cause havoc when viewing the UI object via the Visual Studio designer. For an example of how to work around this problem refer to the sample C# programs supplied with MapLink that utilise the `ControlDesigner` class.

25.3 VB Walkthrough 1 - Your First VB MapLink Application

By the end, you should have an application that can load and display a map generated from MapLink Studio and can correctly handle paint and resize events.

It assumes that you are familiar with both VB and the earlier C++ walk through application.

25.3.1 Skeleton Application

The starting point for this is a VB 'Windows Forms Application' Application Wizard generated executable. These instructions assume that the .NET Framework 4 is targeted.

Use the standard VB 'Windows Application' Application Wizard to generate your skeleton application. The example application here is called "Hello Globe".

25.3.2 Configure Project Properties

Once created, build your skeleton application to ensure it compiles and links. You then need to import the appropriate MapLink .NET libraries into the project references. The MapLink installer does not integrate these .NET assemblies into the Visual Studio standard list of assemblies so you will need to browse to your installations bin directory, E.G.

```
C:\Program Files\Envitia\MapLink Pro\X.Y\bin64
```

Where X.Y is the version of MapLink you are using.

For the purposes of this walk through we will only import the Core MapLink .NET assembly `Envitia.MapLink64.dll`

NOTE: If you look at any of the projects for the C# samples that ship with MapLink you'll find that the MapLink .NET assemblies don't appear under the "References" node of the Solution Explorer. This is due to legacy reasons.

25.3.3 Initialisation and Clean Up

The first thing you'll need to do is import the namespaces to the project for all the newly added MapLink assemblies. This will mean that subsequent references to MapLink classes won't need to be prefixed with the namespaces that contain them. The namespaces for each of the MapLink assemblies are listed in section 25.1.

To import a namespace, navigate to the references tab of the project properties. At the bottom of the page is a list of all the globally imported namespaces.

The configuration files for MapLink are usually only loaded once per execution run using static methods of `TSLNDrawingSurface`. In a VB application this can be done in several places, but the easiest is in the main form's constructor. The simplest way to go about this is to tell MapLink to load all standard configuration files from a particular directory. If no directory is specified, then MapLink will assume that a full MapLink installation has taken place and will attempt to load from there.

Add a constructor to the application's main form and add a call to `TSLNDrawingSurface::loadStandardConfig`. This should be done before the call to `InitializeComponent` in case MapLink classes are constructed during this call.

You should be careful to check for, and report errors at this stage by using the methods supplied on the `TSLNErrorStack` utility class.

```
Public Sub New()
  TSLNErrorStack.clear()
  Dim configDirPath As String = Nothing 'Replace if deployed

  TSLNDrawingSurface.loadStandardConfig(configDirPath)
  Dim msg As String = TSLNErrorStack.errorString(
        "Initialisation Errors : \n",
       TSLNErrorCategory.TSLNErrorCategoryError Or
       TSLNErrorCategory.TSLNErrorCategoryFatal)
  If (Not msg Is Nothing) Then
    MessageBox.Show(Me, msg)
    Environment.Exit(-1)
  End If

  ' This call is required by the Windows Form Designer.
  InitializeComponent()
End Sub
```

When your application is deployed, make `configDirPath` variable point to the location of your applications copy of the MapLink `config` directory.

Once MapLink has been initialised, it needs to be cleaned up when the application exits, otherwise Visual Studio will report numerous "leaks" which are in fact memory currently in

use when the application exits. This should be done in the `Dispose` method of the main form.

> The tidy up of MapLink should occur after the form's components have been disposed of but before the form itself is disposed of.

```
Protected Overrides Sub Dispose(ByVal disposing As Boolean)
  Try
    If disposing AndAlso components IsNot Nothing Then
      components.Dispose()
    End If
  Finally
    TSLNDrawingSurface.cleanup() 'MapLink Code
    MyBase.Dispose(disposing)
  End Try
End Sub
```

25.3.4 The Drawing Surface and Map Data Layer

First of all, we'll add some UI features to the main form to allow users to load a map:

- Add a Menu Strip to the form (Drag from the toolbox to the top of the form in designer mode)
- Add a 'File' menu group.
- Add an 'Open' and 'Exit' menu items to the File menu group. Add an event handler for each of the operations. (Click on the buttons to achieve this)
- Add an `OpenFileDialog` object to the main form. (Drag the icon from the toolbox to anywhere on the form). Set the filter in the properties window to:

```
MapLink Maps|*.map;*.mpc|All files|*.*
```

Also set the title and any other settings if required.

Next we'll declare the variables required and setup the drawing surface:

- Before the constructor in the main form's main class, declare private instances of both the TSLNDrawingSurface and TSLNMapDataLayer class and initialise them to null.
- Add an event handler for the main form's 'Load' event. This can be achieved via the properties window when the form is viewed from the designer by clicking on the events (the lightning icon) buttons at the top. When viewing the form events, find the Load event and type a method name next to it, or double click for the default method name to be used.
- In the load event handler construct the TSLNDrawingSurface passing in the form's 'Handle' member variable and the second argument as false to indicate that the handle is a window handle. For example:

```
m_drawingSurface = New TSLNDrawingSurface(Me.Handle, False)
```

Finally, we'll hook up the menu event handlers to allow the map to load and the program to exit.

- In the File Open event handler call you instance of the `FileOpenDialog's ShowDialog` method. Capture the return value and return from the method if it anything but `DialogResult.OK`.
- Construct your instance of the `TSLNMapDataLayer` class and call its `loadData` method using the filename retrieved from the user.
- Check the return value of the `loadData` call and if it fails check the error stack for the reason.
- Finally add the new data load to your instance of `TSLNDrawingSurface` via the `addDataLayer` method, assigning it a unique name.

For example:

```
Private Sub OpenToolStripMenuItem_Click(...
  If (OpenFileDialog1.ShowDialog() <> DialogResult.OK) Then
    Return
  End If
  m_mapDataLayer = New TSLNMapDataLayer()
  If (Not m_mapDataLayer.loadData(OpenFileDialog1.FileName)) Then
    Dim errorStr As String = TSLNErrorStack.errorString("Errors:",
                             TSLNErrorCategory.TSLNErrorCategoryAll)
    m_mapDataLayer = Nothing
    If (Not errorStr Is Nothing) Then
      MessageBox.Show(Me, "Error", errorStr, MessageBoxButtons.OK)
      Return
    End If
  End If
  m_drawingSurface.addDataLayer(m_mapDataLayer, "map")
  m_drawingSurface.reset(True)
End Sub
```

- Lastly when the File Exit menu button is clicked call the form's `Close` method.

At this point it would be advisable to compile the project to check if any coding errors have occurred so far. Although the program should run, you'll find that it won't do very much as we haven't told MapLink when it needs to draw yet!

25.3.5 Handling Paint and Resize Events

Since MapLink is passive, the application needs to handle relevant events and pass the information onto MapLink. Most applications will only need to handle the window paint and resize events.

A paint event can be triggered for many reasons, some of which will only want to redraw part of the window. Under these circumstances, Windows will set up a Clip Box to define the part that needs redrawing. To improve performance, it is best to only redraw that part. It is most efficient to pass the required Device Unit extent to the Drawing Surface.

After handling a resize event, Windows will usually post a paint message so there is no need to force a redraw in the resize handler. Just changing the window size may distort the aspect ratio of the display, so MapLink can automatically adjust the visible map area to be in sympathy with the aspect ratio of the window. This optional behaviour allows an

anchor point to be specified, which is kept at the same place when updating the visible map area.

To add `Paint` and `Resize` event handlers to your form, the steps are the same as outlined for adding a `Load` event handler detailed earlier in this walkthrough.

To the Paint event handler, first check that the drawing surface has been constructed and if so, request a redraw via the `drawDU` method, e.g.:

```
Private Sub Form1_Paint(...
  If (m_drawingSurface Is Nothing) Then
    Return
  End If

  m_drawingSurface.drawDU(e.ClipRectangle.Left,
                          e.ClipRectangle.Top,
                          e.ClipRectangle.Right,
                          e.ClipRectangle.Bottom,
                          True)
End Sub
```

To the `Resize` event hander, once again first check that the drawing surface has been constructed and if so, request a resize via the `wndResize` method. The second to last argument is whether a redraw should occur, which is required as .NET will only redraw if the control gets larger. The final argument to the `wndResize` method dictates how the existing view should relate to the new view, e.g.:

```
Private Sub Form1_Resize(...
  If (m_drawingSurface Is Nothing) Then
    Return
  End If

  m_drawingSurface.wndResize(ClientRectangle.Left,
                             ClientRectangle.Top,
                             ClientRectangle.Right,
                             ClientRectangle.Bottom,
                             True,
              TSLNResizeActionEnum.TSLNResizeActionMaintainCentre)
End Sub
```

Finally, we need to handle the initial resize of the main window. For some reason a Resize event is not sent by the .NET framework when the form is initially sized. So we'll have to tell the drawing surface its initial size. Simple copy the `wndResize` statement into your `Load` event handler after the drawing surface has been constructed.

Now build the program, run it and load one of the sample maps.

25.3.6 Further Tweaks

One of the problems with the setup used in the walkthrough is that the menu strip at the top of the application hides some of the map area. You'll notice this affect if you load a map as the area of white space at the bottom won't match the amount at the top. In the sample C# applications supplied with MapLink we get around this issue by moving the drawing surface into a panel within the main form's client area. The same effect could,

however, be achieved by sizing the drawing surface using the menu strip's bottom coordinates.

To add double buffering to the form in VB, users will need to override the form/panel's `OnPaintBackground` and not call the base implementation. This will be in addition to calling `setOption` on the drawing surface as described in section 8.10, e.g.:

```
Protected Overrides Sub OnPaintBackground(ByVal pevent As PaintEventArgs)
  ' do nothing...
  ' we don't want the background to flash over the map
End Sub
```

NOTE: Override the `OnPaintBackground` method can cause havoc when viewing the UI object via the Visual Studio designer. For an example of how to work around this problem refer to the sample C# programs supplied with MapLink that utilise the `ControlDesigner` class.

26 FLOATING POINT

The MapLink libraries assume that the floating-point unit is set to that used by the C/C++ runtime.

If you are using MapLink via .NET or similar technologies, you need to ensure that the floating point setup matches that of the C/C++ runtime.

In addition, the DirectX Accelerator SDK requires a specific setup (DirectX requirement).

27 OTHER SDKS

These other MapLink SDKs will be discussed in much greater detail in a future version of this document. Please contact Envitia support to see if there is a newer version available.

The intended topics are:

Interactive Editing with the Editor SDK:

- Editing Architecture
- Interactive Operations
- Entity Creation
- Entity Manipulation

Network Analysis and Routing:

- Network Construction
- Network Traversal
- Routing and Complex Cost Objects

28 THREADING

This section applies to the Core SDK, Terrain SDK, 3D SDK, Accelerator SDK and Dynamic Data Object SDK only. With other SDKs ensure that you do not share objects across threads.

Introducing multi-threading complicates matters as MapLink is not completely thread safe. This is principally to ensure maximum performance.

You should review the whole of this section if you are going to use MapLink in multiple threads.

If you are using MapLink for drawing in multiple threads, then you may need to use the following methods:

- TSLUtilityFunctions::getThreadedOptions
- TSLUtilityFunctions::setThreadedOptions

The following classes have been updated to provide additional functions which do not store results in static data:

- TSLCoordinateConverter
- TSLProfileHelper
- TSLPathList
- TSLCoordinateConverter
- TSLFileLoader
- TSLInteropConfig
- TSLInteropExportSet
- TSLInteropImportSet

The following methods are no longer static:

- TSLMapDataLayer::setRuntimeProjectionParameters (C++ /.NET)
- TSLAPPSymbol::write

28.1 Known Threading Issues

The following are known not to be thread-safe and you should stop all threads before performing any of the following (PLEASE read the following sections as well):

- Any method noted as Deprecated by the compiler should be updated to use the new replacement method.
- Loading and adding `CoordinateSystems` and `CoordinateSystem` registries.
- Static setter methods are not thread safe. For example, you should make sure that if you set your own loader or pathlist on the Drawing Surface that

this occurs before your application starts any threads using MapLink (or stop all threads using MapLink outside of MapLink methods).

- Don't share Display connections between threads on X11. Resources are keyed on the Display.
- Seamless Layer Manager is not thread safe.
- Flashback is not thread safe.
- History is not thread safe.
- DBIF is not thread safe.
- Entity Store SDK is not thread safe.
- S63 SDK is not thread safe when saving data.
- Don't setup a Persistent cache with shared layers.
- Sharing Drawing surfaces between threads is not safe.
- `TSLErrorStack` interface is not thread safe, use the `TSLThreadedErrorStack`.

The following Core SDK methods and classes are also known not to be thread safe. If you need to call the methods from a multi-threaded context protect the calls and copy the results immediately.

- `TSLCompareHelper`
- `TSLVariant::id` (use `getID`)
- `TSLMapDataLayer::getPaletteFilename`
- `TSLMapDataLayer::getPathlistFilename`
- `TSLMapDataLayer::getDetailLayerName`
- `TSLMapDataLayer::getOverviewLayer`
- `TSLMapDataLayer::metadata`
- `TSLErrorStack::first` (use `TSLThreadedErrorStack`)
- `TSLErrorStack::index` (use `TSLThreadedErrorStack`)
- `TSLErrorStack::lastError` (use `TSLThreadedErrorStack`)
- `TSLErrorStack::next` (use `TSLThreadedErrorStack`)
- `TSLErrorStack::previous` (use `TSLThreadedErrorStack`)
- `TSLProfileHelper`
- `TSLInteropConfig::basefilename`
- `TSLInteropConfig::groupingAttribute`
- `TSLInteropExportSet::item`
- `TSLInteropImportSet::item`

- `TSLMapDataLayer::copyRasterFeatures`
- `TSLSeamlessLayerManager::setMapLinkVersion`
- `TSLUtilityFunctions::sav( int arg )`
- `TSLPathList::getMatchingDirectories`

  (use `getMatchingDirectoriesMT`)

28.2 Threading Options

Several threading options can be set or cleared when using MapLink. Currently the only one that you should consider using is the TSLThreadedOptionsRenderingSupport (also see 28.6).

The threading options may be set and queried using the following methods:

- `TSLUtilityFunctions::setThreadedOptions`
- `TSLUtilityFunctions::getThreadedOptions`

28.3 Saving Data

MapLink allows you to save data as the current version or as a previous version.

The setting of the version is not local to a layer but is stored globally, because of this we currently take a global lock to ensure data is written out in the correct version.

28.4 Drawing Surface ID

The drawing surface ID that the user can set is no longer used by MapLink. We now create a unique value internally.

The user ids must be positive values. If the user does not set one the internal unique id is returned as a negative number.

28.5 Drawing Surface Resource Loading

In general, you should setup the Drawing Surface resources before your application goes multi-threaded. The line-styles, fill-styles, fonts and symbols are a shared resource and take time to load. The loading is thread-safe however the propagation of the new resources is lazy and only occurs on a draw.

Note: Delayed loading of resources is not thread-safe.

28.6 Drawing Surface Rendering

While in general the drawing is thread safe you should avoid sharing layers between threads (see 28.8).

If you wish to share the `TSLStandardDataLayer` (see 28.9) between threads then you need to call `TSLUtilityFunctions::setThreadedOptions` to set the bit represented by `TSLThreadedOptionsRenderingSupport`.

28.7 Coordinate System Resource Loading

The loading of the Coordinate System information (`TSLCoordianteSystem::loadCoordinateSystems`) is not thread safe and therefore the coordinate systems should be loaded before the application uses MapLink in a threaded manner. A map loads and creates a coordinate system local to the layer so it is not strictly necessary to load all the Coordinate Systems unless you need to convert between different projections.

28.8 Data Layers

The following types of data layers must not be shared between threads:

- Map Data Layers
- CADRG Data Layers
- Raster Data Layers
- WMS Data Layers
- Filter Data Layers
- All Grid Data Layers
- ECW Layer
- Dynamic Object Data-layer and the Display Objects

Adding or removing a layer from a drawing surface is not thread safe. Stop all drawing surfaces that the layer is to be added to or removed from before adding or removing.

28.9 Standard Data Layer

Sharing the standard 2D data-layer is safe as long as the Drawing Surface IDs are different for each thread (see 28.6).

You must not edit (add or remove entities) the layer unless you stop the drawing in all threads the layer has been added too.

The changing of rendering styles is permitted, though the updating of the drawing maybe delayed.

28.10 Custom Data Layer

It is the responsibility of the developer to ensure that the layer is thread-safe.

If you are adding a Custom 2D Data Layer to an Accelerator or 3D surface, you should note that the layer will be called from a back-ground thread.

28.11 Dynamic Rendering

It is the responsibility of the developer to ensure that the dynamic renderer is thread-safe.

If you are adding a dynamic renderer to an Accelerator or 3D surface then you should note that the layer will be called from a back-ground thread

28.12 TSLPathList

TSLPathList is not thread safe unless the application takes the following measures:

1. Do not use the callback.
2. If you are setting up a pathlist for the Drawing Surfaces to use; Set up the drawing surface pathlist object and add this to the drawing surface before your application starts using multiple threads.
3. If you need to change the drawing surface pathlist or modify it after your application has started its threads; Stop all threads outside of MapLink calls and do the necessary modifications.
4. If you are setting up a pathlist for data layers; Ideally use a separate pathlist per map data-layer (do not share cached layers between drawing surfaces in different threads). If you need to share the pathlist between map data-layers then only modify the pathlist when all threads using MapLink are stopped outside of MapLink method calls.

28.13 User Geometry

It is the developer's responsibility to ensure thread safety for the user implemented functionality

28.14 Dynamic Data Object Layer

The Dynamic Data Object Layer should not be shared between Drawing Surfaces in different threads.

The layer and its associated objects should only be modified from the thread containing the associated Drawing Surface.

28.15 Terrain SDK and Contouring SDK

Terrain can be used in multiple Threads as long as the terrain layer (`TSLTerrainDatabase`) is unique for each thread (not shared between threads).

The Contouring SDK has not been used in a threaded manner and thus may not be thread-safe. The Contouring SDK modifies the floating-point registry to enable strict IEEE floating point and as such is unlikely to be thread-safe.

28.16 3D SDK & Accelerator SDK

Both the 3D and Accelerator Drawing surfaces use a background thread for drawing 2D layers.

The 3D SDK and the Accelerator SDK clone their Map, CADRG and WMS Data Layers to ensure thread safety. These layers can be shared between Drawing Surfaces in the same thread.

Other data-layers are not currently cloned and as such you should not share these layers between multiple drawing surfaces, with the exception of the 2D Standard Data Layer (see 28.9). In addition, you should stop the drawing surfaces before modifying the layers in any-way.

All data layer types, except the S63 and ECW Data-layer, should be acceptable to the 3D SDK and Accelerator SDK. If you need to use these layers with the 3D or Accelerator SDK please contact support.

- Drawing should always occur from the thread that created the Surface.
- Sharing Drawing surfaces between threads is not safe.
- Picking with 3D Surface is not thread safe. Picking must occur in the main drawing thread.
- Removing and deleting 3D entities is not thread safe. Removing and deleting of entities must occur in the main drawing thread.
- Deleting a layer in a thread other than the main drawing thread is not thread safe (deletion of OpenGL/DirectX resources will occur).

28.16.1 Accelerator Drawing Surface Rendering

While in general the drawing is thread safe you should avoid sharing layers between threads (see 28.8).

If you update a layer's content the changes will not be reflected upon the display until you have called notifyChanged on the surface.

28.16.2 3D Drawing Surface Rendering

While in general the drawing is thread safe you should avoid sharing layers between threads (see 28.8).

If you wish to share the `TSL3DStandardDataLayer` between threads, you must call `TSLUtilityFunctions::setThreadedOptions` to set the bit represented by `TSLThreadedOptionsRenderingSupport`. In addition, you must add the layer to all drawing surfaces before you start any drawing and you should not edit the layer once drawing has occurred.

Ideally you should not share the `TSL3DStandardDataLayer` between threads principally because we store data upon the entities which is drawing surface specific and the locking will affect the performance of the drawing.

28.17 X11 Threading

On X11 you must either serialise the calls to MapLink or use a separate display connection for each drawing surface.

Resources are allocated on a display basis and are cached in MapLink based on the Display as the key.

Use of separate display connections in each thread is the safe way to use MapLink. Sharing of Display connections may appear to work until you start using processors with multiple cores or a multi-processor system.

You should call `XInitThreads()` before any other Xlib calls in your application as the Xlib library and generally the extensions are not thread safe until this method has been called. You may need to review the source code of the libraries you use as we know that the Xft extension is not thread safe.

We have found that `XInitThreads()` is not always required if you limit your use to Xlib and avoid Xft (or protected access to Xft methods - see 28.2 and 12.6.9), however this is a case of test and review the client side library source code as the versions you are using may be very different from the ones we have used. Additionally, we have ensured that we do not share X resources between drawing surfaces.

The principle drawing limitation in a threaded environment is the X-Server. The X-Server is a single process so all drawing calls will be serialised at the X-Server. This is not necessarily a problem as MapLink and your application may be able to do something else in the dead time.

Synchronisation calls are kept to a minimum within the X11 Drawing Surface.

App A Developers Guide UNIX/Linux/VxWorks (X11)

MapLink has been ported to a variety of operating systems (OS), the common denominator being that X11 is available for those operating systems.

Limitations which are specific to a particular OS and compiler configuration can be found in the X11 Release Notes.

The principles outlined in the 'Developers Guide for MapLink' are applicable for use with MapLink on an X11 platform. This section covers the differences between Windows and X11 runtime programming.

A.1 Programming for X11

A.1.1 TSLMotifSurface

The primary programming difference between Windows and X11 is the Drawing Surface class that you use. For X11 you will use: `TSLMotifSurface` (should really have been called X11DrawingSurface).

There are two constructors for this class:

- `TSLMotifSurface (Display* display, Screen* screen, Colormap colormap, Drawable handle, int flags = 0, Visual* visual = 0);`
- `TSLMotifSurface (Display* display, Drawable handle, int flags = 0);`

Ideally you should pass as much information to MapLink as possible. This is particularly important when using raster maps, as the actual Visual is required so that the correct raster drawing routines can be used.

`TSLMotifSurface` is a simple class, which provides several additional X11 specific methods, which also have similar methods on the `TSLNTSurface` as follows:

- `int colourValue (int index);`
- `bool fillStyleValue (int index, int colour, Pixmap pixmap, TSLSimpleString *section = 0);`
- `bool fontStyleValue (int index, int colour, Pixmap pixmap, const char** fontName = 0, const char *outputString = 0, TSLSimpleString *section = 0);`
- `bool lineStyleValue (int index, int colour, int thickness, Pixmap pixmap, TSLSimpleString *section = 0);`
- `bool symbolStyleValue (int index, int colour, Pixmap pixmap, uint32_t fontSymbolCharacter = 0, TSLRasterSymbolScalable rasterSymbolScalability = TSLRasterSymbolScalableAsSymbolFile, TSLSimpleString *section = 0);`
- `bool drawToDrawable (Drawable drawable, double x1, double y1, double x2, double y2, bool clear);`
- `bool attach (Drawable handle, bool isPixmap, Display* display = 0, Screen* screen = 0, Colormap colormap = -1, Visual* visual = 0);`

- `bool fullDetach ();`

All the above methods are specific to X11 (not all the default parameters are shown).

When creating an application for X11, regardless of GUI toolkit, the principles behind the 'Walkthrough 1 – Your First MapLink Application' are just as valid. Some samples are included with the CD to help you.

A.1.1.1 Actions on close of Display

You should call `TSLDrawingSurface::cleanup()` before you close the Display.

A.1.2 Using GUI Toolkits with MapLink

MapLink does not depend on any particular GUI toolkit to work. MapLink relies only on Xlib.

It is therefore possible to use MapLink with any number of GUI toolkits, such as Motif or Qt (https://www.qt.io/).

We now ship samples for Qt 4.7 and Qt 5.15.

A.1.2.1 Using Qt4.X

If you have difficulty integrating MapLink with Qt please contact support.

Add a method to the Custom Widget as follows:

```
virtual QPaintEngine *paintEngine() const
{
  return 0;
}
```

This stops Qt drawing into the Widget itself.

In the Custom Widget constructor add the following:

```
// This is required for Qt4 to stop the back ground being drawn
// and Qt Double buffering. You also need to override
// paintEngine().
//
// Ref:
// http://lists.trolltech.com/qt-interest/2006-02/thread00004-0.html
//

setAttribute( Qt::WA_NoBackground, true);
setAttribute( Qt::WA_NoSystemBackground, true);

// Possible issue with this for Qt4.1.0 and newer versions.
//
// See:
// http://www.trolltech.com/developer/task-tracker/index_html?id=106922&method=entry
// http://lists.trolltech.com/qt-interest/2006-05/thread00316-0.html
//
// Talk to Trolltech support about getting a fix if this proves to
// be a problem
//
// NOTE: I am not seeing this problem, probably because I'm doing
// things slightly differently from the example.

setAttribute( Qt::WA_PaintOnScreen, true);

setAutoFillBackground(false); //should be true for Qt4.1 and 4.0
```

For Qt4.1 and newer you will need to add the following:

```
setAttribute(Qt::WA_OpaquePaintEvent);
```

When you construct the MapLink Drawing Surface use the `winId` or `handle` method as follows:

```
// Attaching to the window is much more efficent.
#ifdef WINNT
  HWND hWnd = (HWND) winId();
  m_drawingSurface = new TSLNTSurface( hWnd, false );
#else
  QX11Info x11info = this->x11Info();
  Display *display = x11info.display();
  int screenNum = x11info.screen();
  Visual *visual = (Visual *)x11info.visual();
  Qt::HANDLE colourmap = x11info.colormap();
  Qt::HANDLE drawable = handle();
  Screen *screen = ScreenOfDisplay(display, screenNum);
  m_drawingSurface  = new TSLMotifSurface( display, screen, colourmap,
                                           drawable, 0, visual);
#endif
```

The `paintEvent` in the Custom Widget should look something like this:

```
void MapLinkWidget::paintEvent ( QPaintEvent *rect )
{
  if (m_drawingSurface == NULL)
    create();

  if (m_initialUpdate)
    resizeCanvas();

  const QRect &r = rect->rect();
  long x1 = r.x() ;
  long y2 = r.y() ;
  long x2 = r.x() + r.width() ;
  long y1 = r.y() + r.height() ;
  m_drawingSurface->drawDU( x1, y1, x2, y2, true, true ) ;
}
```

A.1.2.2 Using Qt 5.1 or later

Add a method to the Custom Widget as follows:

```
virtual QPaintEngine *paintEngine() const
{
  return NULL;
}
```

This stops Qt drawing into the Widget itself.

In the Custom Widget constructor add the following:

```
setAttribute( Qt::WA_OpaquePaintEvent );
setAttribute( Qt::WA_PaintOnScreen );
setAttribute( Qt::WA_NativeWindow );
setAutoFillBackground( false );
```

When you construct the MapLink Drawing Surface use the `winId` method as follows:

```
#ifdef WIN32
  m_surface = new TSLNTSurface( (HWND)winId(), false );
```

```
#elif QT_VERSION >= 0x50100
  // Qt 5.1 or newer
  Display *display = QX11Info::display();
  WId wid = winId();

  XWindowAttributes attribs;
  XGetWindowAttributes( display, wid, &attribs );

  m_surface = new TSLMotifSurface( display, attribs.screen, attribs.colormap, wid, 0,
                                   attribs.visual );
#else
  // Qt 5.0 does not provide easy an easy way of accessing the X11 display or drawable
  // for the widget. It is recommended that you upgrade to Qt 5.1 or later.
#endif
```

A.1.2.3 Drawing on top of MapLink using Qt

In order to use Qt to draw on top of MapLink rendering, you will need to draw the map data into a `QtPixmap` and blit the `QtPixmap` to the screen. The code to disable the Qt double buffering and background clearing is probably no-longer required depending on what you are trying to achieve.

A.2 Text Drawing

The X11 drawing code now uses Pango to draw text so that we can support Unicode. On most platforms Pango uses Xft and hence XRender.

On 'Solaris 10 x86' we have had to use the latest Xft because the one shipped is too old to work effectively with XRender. The version of Pango we are using is the latest one that we were able to compile using the development environment available on the platform.

On 'Solaris 10 SPARC' the version of Pango we are using is the latest one that we were able to compile using the development environment available on the platform.

A.3 Dynamic Data Object SDK

Dynamic Data Object (DDO) SDK allows developers to create fully dynamic overlays within a MapLink application (see Developers Guide).

When you create a `TSLDisplayObject` derived class you have two options when implementing the draw method.

1. Make a sequence of calls to the Rendering Interface to set up attributes and draw graphical primitives (portable).
2. Draw using X11 drawing methods (non-portable, optimal).

   Obtaining the `Display` and `Drawable` can be achieved as follows:

```
bool AircraftDO::draw(TSLRenderingInterface *d_surface,
                      TSLEnvelope *d_extent )
{
  long ldisplay;
  Drawable drawable = (Drawable)
              (d_surface->handleToDrawable( &ldisplay ));
  Display* display = (Display*)ldisplay;
  // ……
}
```

The `TSLRenderingInterface` also provides access to the `Visual`, `Colormap` and Screen. This will make it easier to create pixmaps and images in a custom data layer or via a DDO.

A.4 Raster support

All MapLink X11 targets support the display of Raster Maps locally or remotely. The X-Server depth to use (via the Visual) will be dependent on the X-Server and the applications use of colour.

The X11 MapLink runtime supports TIFF, PNG and JPEG formats generated by MapLink Studio (not all combinations of these formats are supported).

Raster datasets from MapLink Studio may be configured to output at a particular bit-depth and for 8-bit images, the number of colours used by the image may be specified. Note: that the default is 24-bit (Please refer to the MapLink Studio help).

Rasters which contain an alpha channel will only be displayed correctly when using X servers that support XRender 0.6 or later.

A.5 Holed Polygons

Vector Maps can be generated from MapLink Studio with holed polygons or without holes by using key-holing.

Using key-holing means that the drawing of holed polygons is a lot less complex and more efficient. Therefore, this is the recommended approach on X11 for performance reasons alone.

The only reason for not using this approach is if a polygon edge line style is required to be displayed as this will show the keyhole construction.

Note: VxWorks target Envitia uses for testing does not support drawing of holed polygons.

A.6 APP-6A and 2525B Symbology

MapLink provides the capability to display many APP6A and 2525B symbols through two classes (`TSLAPP6AHelper & TSLAPP6ASymbol`).

MapLink provides two configuration files, `app6aConfig.csv` and `2525bConfig.csv` in the `config` directory of your MapLink installation. Applications should pass the full path to one of these files depending on which set of symbology is desired.

In addition to these configurations the alternate symbol file '`tslsymbolsAPP6A.dat`' has been provided. This also includes OTHT-Gold symbols. Any application using APP-6A or 2525B icons should load this symbols file. It is acceptable simply to load it using '`setupSymbols`' after calling '`loadStandardConfig`'. See the Windows APP-6A sample.

A.7 Stroked Linestyles

Stroked linestyles are implemented by an extension shared library (`ttlclsstrk.so/sl/o`). The shared library is written specifically for the target platform.

The file `tsllinestyle.dat` contains many stroked linestyle definitions, an example of a stroked linestyle is shown below:

```
1000;ttlclsstrk;My Custom Line;MyCustomLineStyleTag;C[(-1,-1,-1),4]U[0,-
2]D[5,0]D[5,0]BC[(-1,-1,-1),2]U[0,1]D[5,0]D[5,0]D[4,0]
```

The above line is broken down as follows:

```
StyleID;typeOrCustomDLLName;Textual comment displayed in feature book, no
semi-colons allowed;DLL specific information
```

For the `ttlclsstrk.so/sl/dll` (DLL) which implements this type of line, the DLL specific information is:

```
UniqueTag;CommandString
```

The `CommandString` is a chain of

`C[(R,G,B),W]`	colour and width, RGB obvious, W width (width <= 0 is set to 1 pixel). If R, G and B are all -1, then colour defined by feature book is used.
`D[x,y]`	Pen down, line to (currentPositionX + x, currentPositionY + y)
`U[x,y]`	Pen up, move to (currentPositionX + x, currentPositionY + y)
`B`	Bend Point. This is where we can effectively start a new line segment.

Where:

Pen Down means place the drawing point on the paper and draw to the specified position from the current position.
Pen Up means raise the drawing point off the paper and move to the specified position from the current position.
All moves are relative to the current position.

The easiest approach when creating or modifying a Stroked Linestyle is to use a pen and a piece of graph paper, recording exactly how you draw the line (pen up, pen down, colour, amount moved).

So for '`D[5,0]U[5,0]D[5,0]`', you get the following simple line:

```
'-----  -----'
```

Where:

-	represents pen colour being drawn (Pen Down) as a solid line, but here represented as dashes to show the amount of movement of the Pen.
space	represents pen colour not being drawn (Pen Up)

The start point of your line is always at position `[0, 0]`.

In the above simple line at the end of the sequence the current drawing position is `[0, 15]`.

So if you wanted to return to `[0, 0]`, you would add '`U[0,-15]`' to the line definition.

Please note the following:

- When drawing a custom linestyle Maplink uses the horizontal axis where y=0, as the middle of the line.
- Progress has to be made in the x-axis.
- The line thickness is specified in pixels. So a line thickness of three will be drawn in a similar way that Windows/X11 will draw a solid line of thickness 3.
- Line segments are drawn with a round end cap on Windows. On X11 line segments are drawn with `CapButt` and `JoinMiter`.
- You can also increase the number of 'B's to improve the ability of the customline style to follow the draw points.
  In general 'B' points must occur when the y-axis is at 0. If you make changes to a linestyle check the changes using a relatively complex map or drawing.
- Custom linestyles will have an impact on drawing performance. The more complex a linestyle the larger the impact on performance.

A.8 X11 Error Handlers

If you define Error handlers by calling `XSetErrorHandler`, then you need to call any error handlers already defined (`XSetErrorHandler` returns the previous error handler).

The Raster drawing shared library uses this error handler to detect problems when using shared memory (`XShm`). The MapLink3D optional SDK also hooks into the X error handling to figure out if it can use the `XShm` extension.

You should setup the X error handlers before you call MapLink or MapLink will call the default handlers if it does not handle the errors it-self. MapLink will not call any error handlers if it handles the errors itself.

It is possible to disable the setting up of the Error handlers by setting of the environment variable `TTL_GDK_SHM_OFF`.

App B Vector and Raster Data Format Support

The list of data formats supported by MapLink Pro Studio or runtime SDKs is constantly being expanded. The following sections describe some of the formats currently supported at the time of writing.

B.1 Vector Datasets

Data Format	Studio Import	Direct Import SDK	Other Runtime Import	Runtime Export
DAFIF	✓			
DFAD	✓			
DXF	✓	✓		
Envitia ASCII	✓			
File Geodatabase (FileGDB)	✓	✓		
GDF3	✓			
GeoPackage	✓	✓		
GML2/GML3	✓	✓	✓	✓
Jeppesen	✓			
KML Simple Features 2D	✓	✓	✓	
MIF/MID	✓	✓	✓	✓
NITF/NSIF	✓	✓	✓	
OpenStreetMap	✓	✓		
OS MasterMap	✓	✓	✓	✓
OS NTF	✓	✓	✓	✓
OS VectorMap Local	✓	✓	✓	
OS VectorMap District	✓	✓	✓	
OS Boundary Line 2000	✓	✓		
S-57 (Unencrypted ENC & AML)	✓	✓	✓	✓
S-57 Encrypted (S-63)			✓	
ShapeFiles	✓	✓	✓	✓
US Census TIGER/Line	✓	✓		
VPF (DNC, VMAP, WVS etc.)	✓			

Other Vector (e.g. TAB, Spatialite/SQLite)	✓	✓		

B.2 Raster Datasets

Data Format	Studio Import	Direct Import SDK	Other Runtime Import	Runtime Export
ADRG	✓	✓		
ARCS Chart (Unencrypted)	✓			
ASRP	✓	✓		✓
BSB Nautical Chart Format	✓	✓		
CADRG/CIB	✓	✓	✓	✓
CRP	✓			
ECRG	✓	✓		
ECW	✓			
GeoPackage	✓	✓		
Geospatial PDF	✓	✓		
GeoTIFF	✓	✓	✓	
JPEG	✓	✓		
JPEG2000	✓	✓		
MrSID	✓	✓		
NITF/NSIF	✓	✓	✓	
USRP	✓	✓		
Other Raster (e.g. IMG, PNG etc.)	✓	✓		

App C Deprecated SDKs

C.1 3D SDK

Envitia provide an integration to osgEarth, including display of symbology and draping of all MapLink Pro layers.

Please contact support@envitia.com or your sales representative for additional information.

The 3D SDK incorporates the advantages of 3D terrain data with existing MapLink maps to create a fully immersive environment for reviewing and exploring. Built to extend and strengthen the MapLink family of tools, the 3D SDK offers all of the advantages of the other components, but in a 3D environment.

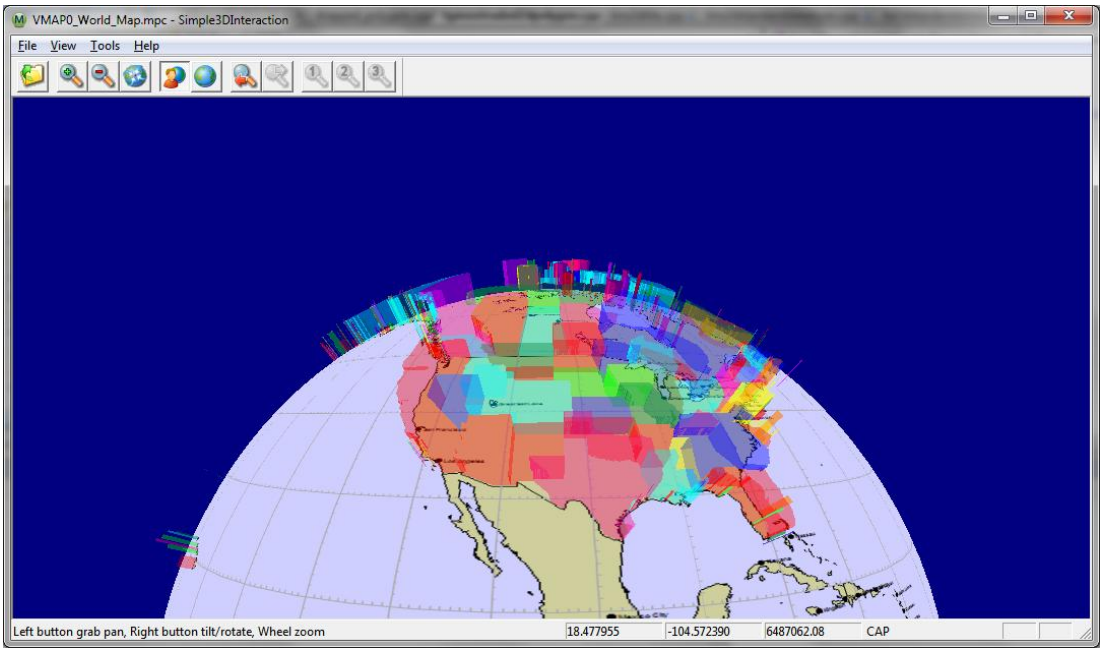


Figure 24 3D Globe with US States Extruded as polygons.

C.1.1 Library Usage and Configuration

MapLink3D64.lib Release mode, DLL version. Uses Multithreaded DLL C++ run-time library. Requires TTLDLL preprocessor directive. Your application must also link the MapLink CoreSDK library MapLink64.lib and the OpenGL library opengl32.lib. Refer to the document "MapLink Pro X.Y: Deployment of End User Applications" for a list of run-time dependencies when redistributing, where X.Y is the version of MapLink you are deploying.

C.1.2 Migrating from 2D to 3D

The MapLink 3D SDK is designed to be completely compatible with the 2D Core SDK and this makes migration very easy. It holds true to many of the Core SDK concepts such as the Document/View model of data being loaded on to a data layer, which are in turn loaded onto a drawing surface.

Another core MapLink concept that is continued in the 3D SDK is the passivity of the library. This greatly increases the flexibility of MapLink but like for the core SDK means that relevant events must be managed by the application and passed onto the 3D drawing surface.

Introduced with this SDK are a number of new data layers, each deriving from the base 3D data layer, `TSL3DDataLayer`. An example of such a layer is the `TSL3DStandardDataLayer` for 3D geometric entities which is the 3D equivalent to the `TSLStandardDataLayer` for 2D geometry. These new data layers, along with any necessary 2D data layers, should be attached to a derivative of the 3D drawing surface base class `TSL3DDrawingSurface`, such as `TSL3DWinGLSurface` for Windows.

C.1.3 The 3D Coordinate Space

All positions in the MapLink 3D world are specified in geodetic coordinates; latitude, longitude and altitude above the surface of the earth. It is possible to perform geodetic to geocentric conversions and the reverse using the `TSL3DDrawingSurface` where the geocentric coordinates x, y, z are from the centre of the earth and their unit is metres. Geodetic coordinates are also wrapped around the earth if they are specified outside of the coordinate space, such as passing over the poles or international data line.

Altitude can also be specified in a number of different ways using the `TSL3DAltitudeType` enum. It can be equated from the mean sea level or from the height above ground level at that point. The height above ground level can also be altered using a range of options to deduce the exact height to use at that point from the terrain data.

It is important to note that as bounding boxes of entities and entity sets are specified in geodetic coordinates, it is difficult to manipulate these with reference to the object they were calculated from. The `TSL3DHelper` class provides several helper functions to manipulate a `TSL3DBoundingBox` object, such as the ability to rotate, scale and translate them.

C.1.4 Threading

The 3D Drawing Surface uses a background thread for rendering of the 2D layers.

As such you should review the contents of section 28, in particular sections 28.8, 28.12 and 28.16.

C.1.5 Walkthrough 5 – Your First 3D Application

If you are familiar with the walkthroughs for the Core SDK then this tutorial might seem basic and could be run through quickly concentrating on the information that appears inside the boxes.

C.1.5.1 Skeleton Application

Please note that the Wizards are not available for Visual Studio 2015, see section Error! Reference source not found.**.**

The starting point for this is an MFC Application Wizard generated executable. It can be either an SDI or MDI application, although MDI is not recommended. The example code here will be based upon an SDI application.

C.1.5.2 Configure Project Properties

Once created, build your skeleton application to ensure it compiles and links. You then need to set up the Project Properties according to the version of the MapLink libraries you wish to use with the corresponding 3D SDK library. These are described in sections 5.1 and C.1.1.

In the x64 Debug configuration make the following checks and modifications to the Project Properties:

In 'C/C++', 'Code Generation' category, check that the run-time library is "`Multi-Threaded Debug DLL`"

In 'C/C++', 'General' category, add the MapLink `include` directory as an additional include path, e.g.

```
"C:\Program Files\Envitia\MapLink Pro\X.Y\include"
```

In the 'Linker', 'Input' category, add `MapLink64d.lib` and `MapLink3D64d.lib` as object/library modules and in the 'Linker', 'General' category add the MapLink `lib64` directory as an additional library path, e.g.

```
"C:\Program Files\Envitia\MapLink Pro\X.Y\lib64"
```

Make the same changes to the x64 Release configuration, except link against `MapLink64.lib and MapLink3D64.lib` instead.

Add `#include "MapLink.h"` and `#include "MapLink3D.h"` to relevant files. In this example, just add it into `stdafx.h` to keep things simple.

Note: X.Y is the version of MapLink you are using.

C.1.5.3 Initialisation and Clean Up

The configuration files for MapLink are usually only loaded once per execution run using static methods of `TSLDrawingSurface`. In an MFC application, these are normally loaded during the `InitInstance` method of the Application object. The simplest way is to tell MapLink to load all standard configuration files from a particular directory. If no directory is specified, then MapLink will assume that a full MapLink installation has taken place and will attempt to load from there.

In the `InitInstance` method of the App object, add a call to `TSLDrawingSurface::loadStandardConfig`. This should be done before the Document Template is instantiated.

You should be careful to check for, and report errors at this stage by using the methods supplied on the `TSLThreadedErrorStack` utility class.

```
const char * configDirPath = NULL ; // Replace if deployed
// Full path and filename to the file tsltransforms.dat
const char * transformsFile = NULL; // Replace if deployed

TSLThreadedErrorStack::clear() ;

TSLDrawingSurface::loadStandardConfig( configDirPath ) ;

// Required for draped polygons.
TSLCoordinateSystem::loadCoordinateSystems( transformsFile );

TSLSimpleString msg( "" );
bool anyErrors = TSLThreadedErrorStack::errorString( msg,
                                    "Initialisation Errors : \n" ) ;
if ( anyErrors )
{
  AfxMessageBox( msg, MB_OK ) ;
  exit( 0 ) ;
}
```

When your application is deployed, make `configDirPath` variable point to the location of your applications copy of the MapLink `config` directory. The `transformsFile` will need to be handled in a similar manner.

Once MapLink has been initialised, it needs to be cleaned up when the application exits, otherwise Visual Studio will report numerous "leaks" which are in fact memory currently in use when the application exits. This should be done in the `ExitInstance` method of the App class. You will need to use the class Properties Overrides to add this method since the MFC Application Wizard doesn't add it by default. Alternatively, in Single Document applications, it may be called in the destructor of the View or Document class.

Use Properties, Overrides to create an `ExitInstance` method on the App object. In this method, call MapLink to cleanup the configuration file load.

```
int CHelloGlobe::ExitInstance()
{
  TSLDrawingSurface::cleanup( ) ;
  return CWinApp::ExitInstance();
}
```

If you are using the DLL versions of the MapLink libraries, please note the discussion of memory leaks in section **Error! Reference source not found.**.

C.1.5.4 Managing the Document

In terms of the Document/View architecture, the Document contains one or more MapLink Data Layers. This is where using the 3D SDK differs greatly from the 2D, for it offers a number of new Data Layers. For the purposes of this example application however, we shall restrict this to a single `TSLMapDataLayer`.

In the private section of the Document, declare a `bool` and a pointer to a `TSLMapDataLayer` object. The `bool` should be constructed to `false`, and the object should be constructed in the document constructor and then destroyed in the destructor:

```
CHelloGlobeDoc::CHelloGlobeDoc () : m_newMap( false )
{
  m_mapDataLayer = new TSLMapDataLayer() ;
}
CHelloGlobeDoc::~CHelloGlobeDoc ()
{
  if ( m_mapDataLayer )
  {
     m_mapDataLayer->destroy() ;
     m_mapDataLayer = NULL ;
  }
}
```

Use Properties, Overrides to create an `OnOpenDocument` handler and in this method, set you `bool` flag to `true` and store the filename in a member variable. Create a private method `loadMap` that takes no parameters and returns `void`

```
BOOL MapLink3DSimpleDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
  if (!CDocument::OnOpenDocument(lpszPathName)) return FALSE;

  m_newMap = true ;
  m_mapName = lpszPathName ;
  return TRUE;
}


void MapLink3DSimpleDoc::loadMap()
{
  if (!m_newMap) return ;

  m_mapDataLayer->removeData() ;
  TSLThreadedErrorStack::clear() ;
  // Load map and then display any errors that have occurred
  m_mapDataLayer->loadData( m_mapName.c_str() ) ;
  TSLSimpleString msg( "" );
  bool anyErrors = TSLThreadedErrorStack::errorString( msg,
      "Cannot load map\n" ) ;
  if ( msg )
    AfxMessageBox( msg, MB_ICONERROR ) ;
  else
    m_mapDataLayer->notifyChanged() ;
}
```

C.1.5.5 Managing the View

In terms of the Document/View architecture, the View contains an instance of a `TSL3DDrawingSurface` derived object – `TSL3DWinGLSurface` on Windows platforms, `TSL3DX11GLSurface` on X11 platforms. This is the only significant platform-specific difference. In an MFC application, this is usually instantiated in the `OnInitialUpdate`

method since the associated window doesn't exist in the `OnCreate` event or in the View constructor.

In the private section of the View, declare a pointer to a `TSL3DWinGLSurface` object. This should be initialised to `NULL` in the View constructor.

Use Properties, Overrides to create an `OnInitialUpdate` handler and in this method, check to see if a Drawing Surface exists and create one if necessary. You can optionally also set a sky, wire frame and solid colours as well as drape a picture over the earth, as is done below. You will need a private member variable of type `CString`, called `m_backdrop` to allow you to do this. You should also tell MapLink about the default size of the window.

```
void CHelloGlobeView::OnInitialUpdate()
{
  CView::OnInitialUpdate();

  if ( !m_drawingSurface )
  {
    CRect rect ;
    GetClientRect( &rect ) ;

    // Create the drawing surface
    m_drawingSurface = new TSL3DWinGLSurface ( m_hWnd, false );

    // Give the 'sky' a colour!
    static const TSLStyleID skyColourIndex( 4 );
    m_drawingSurface->setBackgroundColour( skyColourIndex );

    static int const wireframeColourIndex( 181 );
    static int const solidColourIndex( 60 );

    m_backdrop = TSLUtilityFunctions::getMapLinkHome();
    m_backdrop += "/config/earth.png";

    // Set the bitmap to display over the terrain plus colours
    // for solid-backdrop and wireframe rendering.
    m_drawingSurface->setTerrainRendering( wireframeColourIndex,
                                           solidColourIndex, m_backdrop );
    // Notify surface what size the window is
    m_drawingSurface->wndResize(0, 0, rect.Width(), rect.Height());

    // The following line is discussed in 12.5.9
    m_drawingSurface->setRenderingCallback(renderingCallback, this);
  }
}
```

In the destructor of the View, destroy the Drawing Surface if it exists.

```
CHelloGlobeView::~CHelloGlobeView()
{
  if ( m_drawingSurface )
  {
    m_drawingSurface->destroy() ;
    m_drawingSurface = NULL ;
  }
}
```

C.1.5.6 Binding Layers and Drawing Surfaces

Once both Document and View are ready available, you need to attach the Data Layers to the Drawing Surface so that MapLink can display it.

The recommended approach to this is to create an `addToSurface` method on the Document, which calls the underlying MapLink routines to add the Document's Data Layers to the Views Drawing Surface. This structure avoids the View knowing the contents of Document in any detail and is equally applicable to both Single and Multiple Document Interfaces.

The `addToSurface` method should be called in the `OnInitialUpdate` method of the View, just after the Drawing Surface has been created. In MFC applications, it is not usually necessary to have an equivalent `deleteFromSurface` method since MFC calls `DeleteContents` instead. If you are adding more than one Data Layer to the Drawing Surface, each must have a unique name.

Create a public `addToSurface` method in the Document that takes a `TSLDrawingSurface` pointer as a parameter. In this, add the Document's Data Layer to the specified Drawing Surface.

```
bool CHelloGlobeDoc::addToSurface(TSL3DWinGLSurface *drawingSurface)
{
  if ( !m_mapDataLayer || !drawingSurface )
    return false ;
  loadMap(); // load the map.

  return drawingSurface->addDataLayer( m_mapDataLayer, "map" ) ;
}
```

Call this method in the View's `OnInitialUpdate` method, after the Drawing Surface has been created. At this point, it is also appropriate to define the initial visible area. Here we call the `reset` method of the `TSL3DCamera`, before providing a position for the camera and the direction in which it is pointing. The workings of `TSL3DCamera` are discussed in 12.8.

```
if ( GetDocument()->addToSurface( m_drawingSurface ) )
{
  m_drawingSurface->camera()->reset();
  m_drawingSurface->camera()->moveTo( 50.0, 0.0, 10000000.0,
  TSL3DCameraMoveActionNone ) ;
  m_drawingSurface->camera()->lookAt( 50.0, -5.0, 0.0, false ) ;
}
```

Note that MapLink automatically takes care of Data Layer and Drawing Surface separation when either is destroyed.

C.1.5.7 Handling Resize Events

Since MapLink is passive, the application needs to handle relevant events and pass the information onto MapLink. Most applications will only need to handle the window resize and expose or paint events.

After handling a resize event, Windows or X will usually post a paint message so there is no need to force a redraw in the resize handler.

Use Properties, Messages to create a `WM_SIZE` handler on the View class since it is not there by default. In this method, check to see if a Drawing Surface exists and if so, pass the new corners of the window to the Drawing Surface using the `wndResize` method. This example will also inhibit an automatic redraw and ask MapLink to maintain the aspect ratio locking the top left corner of the visible map area.

```
void CHelloGlobeView::OnSize(UINT nType, int cx, int cy)
{
  CView::OnSize(nType, cx, cy);

  if ( m_drawingSurface )
  {
    m_drawingSurface->wndResize( 0, 0, cx, cy, false );
  }
}
```

Handling resize events differs from the 2D to the 3D as we are not given the option of providing a flag to indicate an anchor point that the resizing takes place around. This is because the `TSL3DCamera` takes care of this control and is discussed in section 12.8.

C.1.5.8 Handling Paint Events

A paint event can be triggered for many reasons, some of which will only want to redraw part of the window. Under these circumstances, Windows will set up a Clip Box to define the part that needs redrawing. To improve performance it is best to only redraw that part. It is most efficient to pass the required Device Unit extent to the Drawing Surface.

In the `OnDraw` method of the View, query the required redraw area and pass it to the Drawing Surface, asking MapLink to clear the background first.

```
void CHelloGlobeView::OnDraw(CDC* pDC)
{
 if ( m_drawingSurface )
 {
  RECT rect ;

  if ( pDC->GetClipBox( &rect ) == NULLREGION )
   GetClientRect( &rect ) ;
```

To create a 3D application you must also provide a `TSL3DRenderingCallback` triggered when draped data is ready to be rendered. This is a static method that returns a void and takes a void*.

Create a new static method in the View with the following implementation:

```
void Simple3DInteractionView::renderingCallback(void * arg,
                                                int pendingTextures )
{
  Simple3DInteractionView * view = (Simple3DInteractionView *)arg ;

  if ( view->m_hWnd )
  {
    view->Invalidate() ;
  }
}
```

Now build the program, run it and load one of the sample maps.

C.1.5.9 Reducing Flicker and Improving Performance

So far, the application is not making use of MapLink performance optimisations and the display will appear to flicker when it is redrawn. There are two reasons for this. Firstly, MapLink is drawing directly to the window. Secondly, both MapLink and Windows are clearing the display prior to the redraw. In depth discussion of these problems and their solutions may be found in section 12.5. In the meantime, here are a couple of quick fixes to reduce your eyestrain! Please be aware this will only work for SDI applications and not for MDI applications.

To solve the first issue, a single method call should be added when the Drawing Surface is created to make it buffered. This will also improve performance on expose events that are not due to the visible map area changing.

To solve the second issue, you should inhibit Windows from clearing the window.

Use Properties, Messages to add a View handler for the `WM_ERASEBKGND` message. Return `TRUE` from this method to indicate to windows that the application will erase the background.

```
BOOL CHelloGlobeView::OnEraseBkgnd(CDC* pDC)
{
  return TRUE ;
}
```

The inhibition of the `WM_ERASEBKGND` message is appropriate since MapLink is drawing to the entire window. If MapLink were drawing to only part of the window then it may be necessary for the application to erase the areas that MapLink is not rendering into.

C.1.5.10 3D Standard Data Layers

The `TSL3DStandardDataLayer` class is a Data Layer, just like the other derivatives of `TSLDataLayer` that have been discussed in this developer guide such as the 2D equivalent `TSLStandardDataLayer`. As such, it may be created and added to one or more Drawing Surfaces from whence the contents are displayed. It is a specialist data layer for the handling of non-map 3D data, providing the ability to load, create, manipulate and save non-map 3D data as well as a number of miscellaneous functions.

In the same way that a `TSLStandardDataLayer` contains instances of `TSLEntity` derived objects and the `TSLObjectDataLayer` contains instances of `TSLDynamicDataObject` derived objects, the `TSL3DStandardDataLayer` contains instances of `TSL3DEntity` derived objects.

C.1.6 3D Entities

A further 2D Core SDK concept that has been continued in the 3D SDK is the use of geometry Entities. All geometric objects in MapLink can be thought of as Entities, derivatives of `TSLEntity` in the 2D and of `TSL3DEntity` in the 3D.

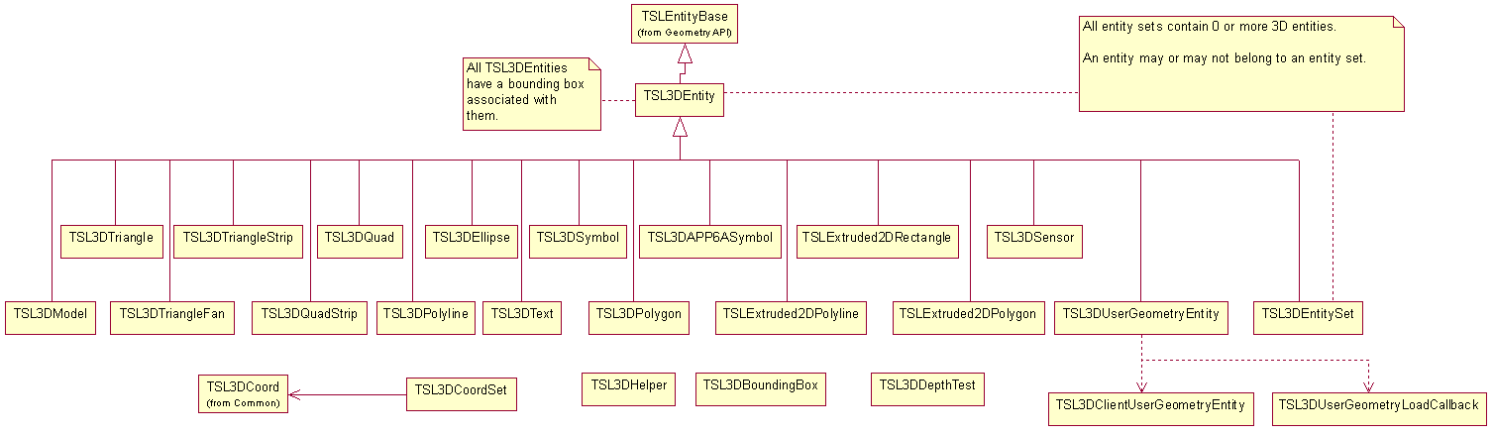


Figure 25 3D Entity Hierarchy

In the 3D SDK all entities with the exception of user geometry have a number of properties including:

- A bounding box defined in 3D space
- A set of rendering attributes that specify how the entity appears.
- One or more `TSL3DCoord` objects that define the position and in most cases the orientation and size of the entity.

C.1.6.1 TSL3DEntity

This is the base class for all 3D geometric primitives and gives access to the methods and properties common to all its derivatives. These include the ability to query the type of derivative an entity is, the bounding box, the centre of the object and the distance this entity is from a specific point. Other operations perform movement and scaling functions and equality comparisons.

C.1.6.2 TSL3DModel

This class defines a common interface to 3D models that can be loaded via plug-ins. The model to draw is determined by setting the `TSLRenderingAttributeModelStyle` rendering attribute to an index from `tslmodels.dat`.

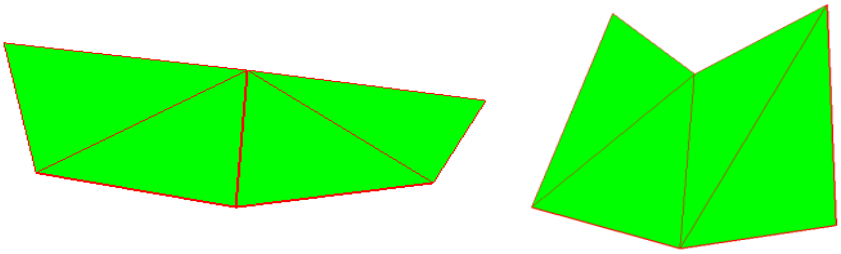
Multiple Levels of Detail can be set for a model to allow for progressively lower polygon-count models to be used when the model is further away from the camera.

C.1.6.3 TSL3DTriangle and TSL3DQuad

Both of these shapes are basically restricted types of polygon; they are limited to having 3 or 4 point and may not have inners. They can be created like the other multipoint 3D entities by passing a `TSL3DCoordSet` or uniquely they can be created by passing the individual `TSL3DCoord` objects. Also like other multipoint geometric shapes, their area and perimeter can be queried. A quad specifically must be non-complex, meaning there are no intersecting edges, and all points must lie in a plane.

The order of point specification is anti-clockwise.

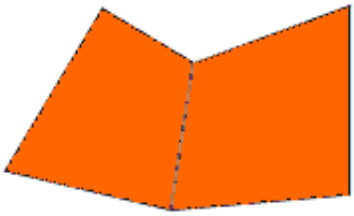
C.1.6.4 TSL3DTriangleFan and TSL3DTriangleStrip



These 3D geometric objects provide a quick way of creating multiple adjoining triangles that use the same rendering attributes. Both are created from closed, filled, 3 point triangles and behave similarly.

For a `TSL3DTriangleFan`, the first point defines the common centre point of the fan. The first three points of the fan define a 3D triangle. Each subsequent point defines a triangle made up of the common centre point, the previous point and the new point. For a `TSL3DTriangleStrip`, the first three points of the strip also define a 3D triangle. Each subsequent point defines a triangle made up of the new point and the previous two points

C.1.6.5 TSL3DQuadStrip



This is the 4 point version of `TSL3DTriangleStrip` and is formed in much the same way; each pair of added points forms a quad with the previous pair. Each contained 3D quad must be non-complex. All points of each contained 3D quad must lie in a plane.

C.1.6.6 TSL3DPolyline

This is the 3D version of `TSLPolyline` which always has length and may or may not have area depending upon whether the polyline is closed. If a polyline is closed then the first and last points are joined by a vertex, except if they exist at the same 3D position in which case the polyline is already closed. A closed polyline can therefore be thought of as being a polygon; the length property becomes the equivalent to its perimeter and it now has the concept of area.

A polyline must have at least two points, although a closed polyline should logically have at least three, but other than that there are no limitations placed upon the coordinates.

C.1.6.7 TSL3DPolygon

A `TSL3DPolygon` is a closed, filled, planar feature with three or more constituent points. It always has a perimeter length property and an area, but due to it being planar it can never have volume. It must be non-complex, meaning its edges must not cross, although they may touch.

A 3D polygon may also have one or more holes also known as inners, with the main polygon also being known as the outer. These inners are basically cut out sections of the polygon which may not touch or cross the outer, nor touch or cross any other hole. The outer nor inners may have consecutive duplicate points.

Draped Polygons, including extruded, have a number of limitations. The applicable limitations are listed in the Release Notes.

For draped polygons to work `TSLCoordinateSystem::loadCoordinateSystems()` must be called before the 3D SDK is used.

C.1.6.8 Extruded 2D Primitives

These extruded shapes, `TSLExtuded2DPolygon`, `TSLExtruded2DPolyline` and `TSLExtruded2DRectangle` consist of a MapLink 2D shape that has been given an extrusion and placed at a set altitude in a 3D world. They are created around the 3D shape, which can be queried or changed for another without destroying the extruded shape. These shapes have identical properties to their 2D counterparts, most of which are accessible by first querying this object for its 2D object.

C.1.6.9 TSL3DEntitySet

This is a collection of other 3D Entities, but is also an entity itself so can contain other Entity Sets and thus be hierarchical. It has no geometric attributes of its own, but inherits its bounding box as the union of its children's. Like the 2D version of this object, the `TSL3DEntitySet` differs from the OpenGIS specification of an entity collection by allowing different types of entity to be contained.

C.1.6.10 3D User Geometry

This is the 3D version of user geometry.

A 3D user geometry entity allows the user to create custom-drawn geometry upon 3D standard data layers. User geometry can be saved to and loaded from TMF files. A piece of 3D user geometry is composed of two parts, the entity (an instance of `TSL3DUserGeometryEntity`, managed by MapLink) and the client (an instance derived from `TSL3DClientUserGeometryEntity`, managed by the user).

C.1.6.11 TSL3DUserGeometryEntity

This is the 3D version of `TSLUserGeometryEntity`.

Instances of `TSL3DUserGeometryEntity` can be added to 3D standard data layers, and are allocated and deallocated by MapLink. Create instances by calling `TSL3DUserGeometryEntity::create`, or by calling `create3DUserGeometry` on a `TSL3DEntitySet`. The client of a 3D user geometry entity can be set and retrieved by calling `setClientUserGeometryEntity` and `getClientUserGeometryEntity`, respectively.

`create`, `create3DUserGeometry`, `setClientUserGeometryEntity` and load callback functions all provide a `takeOwnership` flag. If true, then MapLink will automatically `delete` the client if it is replaced with `setClientUserGeometryEntity` or when the entity is destroyed. If false, the user will have to destroy the client. This must be false if the user's code is compiled with a different compiler or runtime library version to MapLink.

Creating and destroying user geometry:

```
TSL3DStandardDataLayer* stdLayer = ...;
TSL3DClientUserGeometryEntity* client = new ...;

TSL3DUserGeometryEntity* entity = stdLayer->entitySet()->
                                        create3DUserGeometry(client, false);
if (!entity)
  ... // handle error

...

entity->destroy();
delete client;   // don't need this if takesOwnership is true
```

C.1.6.12 TSL3DClientUserGeometryEntity

This is the 3D version of `TSLClientUserGeometryEntity`.

The user creates clients by deriving from `TSL3DClientUserGeometryEntity`, and creating their own instances of these subclasses. A client can then be attached to an entity as explained above.

At a minimum, the user must override the abstract `draw` and `centre` methods. It is however strongly recommended that `boundingSphereRadius` is also implemented within the client. In conjunction with the position returned from `centre`, the return value from `boundingSphereRadius` is used to perform view frustum culling of user geometry. The user should therefore ensure that the calculated bounding sphere radius is accurate for the entity being rendered in order to avoid user entities that are visible from being incorrectly culled.

Unlike 2D geometry, view frustum culling is performed on `TSL3DEntitySets` as well as individual entities. If the size of the user geometry changes it is necessary to manually update the bounding boxes of its parent `TSL3DEntitySet` as the previous bounding box may no longer be correct. This is done by calling `updateBoundingBox` on the TSL3DEntitySet that contains the `TSLUserGeometryEntity` object associated with the client. This entity set can easily be retrieved by using the `parent` method of the `TSLUserGeometryEntity`.

Within the `draw` function, the entity will be positioned such that (0,0,0) in model space is at the location returned by the user's `centre` method with the positive Z-axis perpendicular to the surface of the earth (ignoring terrain) at that point. This means that within a draw each user geometry object operates within its own local coordinate system, the units of which are metres. The exception to this is any drawing performed through methods on the `TSL3DRenderingInterface` that accept positions using `TSL3DCoords`, `TSL3DCoordSets` or a `TSL3DEntity`. Objects rendered in this fashion are drawn in the same positions as they would be if drawn from outside user geometry.

The OpenGL state on entry to `draw` is dependent on the entities that have been drawn so far in the current frame, and therefore will differ depending on the view of the application. The user should therefore make no assumptions about the OpenGL state on entry to `draw` other than the following:

- The `GL_COLOR_ARRAY`, `GL_EDGE_FLAG_ARRAY`, `GL_FOG_COORD_ARRAY`, `GL_INDEX_ARRAY` and `GL_SECONDARY_COLOR_ARRAY` client states will never be enabled.
- The matrix mode for the built-in matrix stack will be `GL_MODELVIEW_MATRIX`.
- The active texture unit will be `GL_TEXTURE0`, however texturing may be either enabled or disabled.
- There will be no active program bound.

MapLink internally tracks the OpenGL state in order to avoid redundant state changes. Therefore care should be taken to reverse any modifications made to the OpenGL state in before returning from `draw` as failure to do so may result in incorrect rendering of subsequent entities. This also applies to any rendering performed through the `TSL3DRenderingInterface`. Aside from this restriction the user is free to use any OpenGL functionality within `draw` in order to render the entity.

Here is an example partial implementation of a user geometry client:

```
class SquareClient : public TSL3DClientUserGeometryEntity
{
private:
  TSL3DCoord m_centre;
  double m_radius;

public:
  // Constructor
  SquareClient(TSL3DCoord centre)
    : m_centre(centre)
    , m_radius(sqrt(2000000.0*2000000.0 + 2000000.0*2000000.0))
  {
  }

  // Destructor
  virtual ~SquareClient()
  {
  }

  virtual double boundingSphereRadius () const
  {
    return m_radius;
  }

  virtual const TSL3DCoord& centre () const
  {
    return m_centre;
  }

  // render an orange square
  virtual bool draw (int uniqueSurfaceID,
                     TSL3DRenderingInterface* renderingInterface)
  {
    glPushAttrib( GL_ALL_ATTRIB_BITS );
    glPushClientAttrib( GL_CLIENT_ALL_ATTRIB_BITS );

    GLfloat coords[] = { -100000.0f, -100000.0f, 0.0f,
                         100000.0f, -100000.0f, 0.0f,
                         -100000.0f, 100000.0f, 0.0f,
                         100000.0f, 100000.0f, 0.0f };

    glColor4f( 1.0f, 0.5f, 0.0f, 1.0f );

    glDisable( GL_TEXTURE_2D );
    glDisable( GL_CULL_FACE );
    glEnableClientState( GL_VERTEX_ARRAY );
    glDisableClientState( GL_TEXTURE_COORD_ARRAY );
    glDisableClientState( GL_INDEX_ARRAY );

    glVertexPointer( 3, GL_FLOAT, 3 * sizeof( GLfloat ), coords );
    glDrawArrays( GL_TRIANGLE_STRIP, 0, 4 );

    glPopAttrib();
    glPopClientAttrib();
    return true;
  }

  // stream out the polygon
  virtual int save (TSLofstream& stream)
  {
    ...
    return SQUARE_USER_GEOMETRY_ID;
  }

};
```

C.1.6.13 Loading and saving 3D user geometry

The process is almost identical to that of 2D user geometry.

If the user wants their 3D user geometry classes to be saved and loaded along with other types of geometry, they need to override the `save` method on the client, and to provide a load callback function to the static method `TSL3DUserGeometryEntity::registerUserGeometryClientLoadCallback`.

The `save` method on the client should return a positive integer to identify the type of 3D user geometry. These numbers should be unique as they can be passed to any registered load callback function. It is suggested that the user publish and track these identifiers.

It is also suggested that the user saves, along with any geometry data, a company identifier, a byte-order mark, a geometry type ID and a version number.

To register a load callback function, a pointer to it must be passed to `TSL3DUserGeometryEntity::registerUserGeometryClientLoadCallback`. The pointer should have type `TSL3DUserGeometryLoadCallback` (which is a function pointer typedef). The pointer will be added to a list; when user geometry is loaded, each function on the list will be called until one returns non-NULL.

Setting a load callback function:

```
TSL3DUserGeometryEntity::
          registerUserGeometryClientLoadCallback(loadUserGeometryCallback);
```

Here is a skeleton load callback function:

```
static TSL3DClientUserGeometryEntity* loadUserGeometryCallback(
                                                    TSLifstream& stream,
                                                    int userGeometryID,
                                                    bool& assumeOwnership)
{
  // whether returned entities will be freed by MapLink:
  assumeOwnership = ...;

  switch (userGeometryID)
  {
    case SQUARE_USER_GEOMETRY_ID:
      ... // stream in client and return it

    ... // etc

    default:
      return NULL;
  }
}
```

C.1.7 3D Custom Data Layers

It is possible to introduce your own custom drawn data to the MapLink 3D drawing surface using the `TSL3DCustomDataLayer` class. To accomplish this you must add an instance of this class to your drawing surface and attach to it your own derivative of the abstract class `TSL3DClientCustomDataLayer`.

Your derivative of the `TSL3DClientCustomDataLayer` class will need to override the pure virtual draw methods, which provide an interface class through which a number of useful functions such as querying if a point or bounding box falls within the viewing volume and coordinate conversion functions can be performed.

C.1.8 Using the Camera

The `TSL3DCamera` class provides the ability to manipulate the users' view of the drawing surface. It has three main properties: its position, orientation and normal. The orientation is also known as its `lookAt` position whereas the normal is perpendicular to this and defines the direction from the centre to the top of the view.

The camera also provides the ability to specify its angle of view, also known as its field of view. This is by default 45 degrees. The camera also allows the user to set the altitude at which the horizon will appear horizontal in the field of view. This altitude should be set to a value at which the horizon has a meaningful definition (e.g. 1000 metres).

C.1.9 Integration with Other OpenGL Applications

It is sometimes desirable to use MapLink in conjunction with user interface toolkits or other libraries that perform their own OpenGL context creation. Depending on the constructor used, the MapLink 3D drawing surfaces can either create their own OpenGL context or use an existing context created externally. When using a drawing surface in this fashion, MapLink can be instructed not to perform buffer swaps through the swapBuffersManually constructor argument, leaving the application in control of when this occurs.
More information can be found in the API documentation for each platform's drawing surface (`TSLWinGLSurface` for Windows, `TSL3DX11GLSurface` for X11 systems).

C.1.10 Creating a 3D Model Plug-in

MapLink provides an example plug-in named `ttl3DS` which is capable of loading files produced by 3D Studio Max. The source code to this plug-in can be found in the `Samples` directory of your MapLink installation.

Model plug-ins are loaded at runtime as models that use them are drawn, and are unloaded when those models are deleted. The plug-in that is used to load and draw a particular model is defined by the `tslmodels.dat` file that is passed to `TSL3DDrawingSurface::setupModels()`. A part of the entry for each model is a plug-in specific string which allows for custom options to be defined for each model and plug-in. New models should be added to this file and given the next available unique index. The count of the number of entries in the file should also be updated. A complete description of the format of this file can be found in the `tslmodels.dat` file provided in the `config` directory of your MapLink installation.

C.1.10.1 The Structure of a Plug-in

All plug-ins must be compiled as DLL/shared objects, and must declare a class that inherits from `TSL3DCustomModel`. An instance of this class will be created for each unique model defined in `tslmodels.dat` that uses this plug-in. In addition to this the DLL/shared object must export the following "C" methods:

```
extern "C" __declspec(dllexport)
     void* getModel( int index,
                     const char* filename,
                     const char* pluginString );

extern "C" __declspec(dllexport) void deleteModel( void* model );
```

When a model is required the `getModel()` method will be invoked with the index from `tslmodels.dat` of the model, the full path to the model file and the plug-in specific configuration string. This method will only be invoked once for each model, and should return an instance of your derived `TSL3DCustomModel` class that is responsible for drawing this model.

When a model is no longer required the `deleteModel()` method will be invoked, with the object returned from the relevant `getModel()` call passed in as the parameter for cleanup by the plug-in.

C.1.10.2 Drawing a Model

A plug-in cannot make any assumptions about the state of the rendering engine when drawing, and should always reset any state changes it makes back to what they were originally before the `draw()` method returns.

Storing and resetting rendering state information in OpenGL.

```
bool N3DSModel::draw(int drawingSurfaceId, double distanceToEye,
                     int lodToDraw)
{
  glPushAttrib( GL_ALL_ATTRIB_BITS );

  // Change any required states and draw the model

  glPopAttrib();

  return true;
}
```

The model itself should be drawn around 0,0,0 and will be translated to the correct position by the 3D SDK. Since the `draw()` method will be invoked frequently for models that are visible in the application the plug-in should make use of optimisation techniques such as display lists to ensure that the drawing takes as little time as possible.

Any textures associated with a model can be loaded via the `TSL3DTextureLoader` utility class. The `loadTexture()` method available on this class returns the texture in a format suitable for passing to the appropriate texture functions used by the type of plug-in, for

example `glTexSubImage2D()` for OpenGL. If the requested texture size differs from the actual size of the texture it will be resized to satisfy the request. For more information see the API class documentation.

C.2 Contouring

The Terrain SDK also allows for the generation of contour lines or polygons from the same height information used in a terrain database. The format that the generated contour information is displayed in is controlled entirely by the application via the use of rendering callbacks.

C.2.1 Providing Data for Contouring

The data to contour is expected in the form of a `TSLTerrainContourVertexList` of `TSLTerrainContourVertex` objects. Each vertex object represents data at a single point, and when all vertices are combined, they should form a regular or irregular grid inside the list object.

Each vertex can store one or more pieces of height information, named 'attributes', for the point it represents. Each of these attributes can be used to model different information about the point that the vertex represents. For example, the first attribute might be height information for the terrain at that point, a second attribute might be a recorded temperature value at that point and a third attribute might be a humidity value. Contour information can be generated separately for each of these attributes. Each vertex within the list must have the same number of attributes.

This example shows loading of height information from a terrain database and storing the data in a `TSLTerrainContourVertexList` ready for the generation of contour lines.

```
// Process the terrain data into a terrain database
if( m_terrainDB.open( terrainDBFile.c_str() ) != TSLTerrain_OK )
  return false;

// Query the extent of the terrain data
long x1, y1, x2, y2;
if( m_terrainDB.queryExtent( x1, y1, x2, y2 ) != TSLTerrain_OK )
  return false;

// Inform the terrain database of the size of our drawing surface
// so it can determine a good resolution for the data
long duMinX, duMaxX, duMinY, duMaxY;
m_drawingSurface->getDUExtent( &duMinX, &duMinY, &duMaxX, &duMaxY );
m_terrainDB.displayExtent( duMaxX - duMinX, duMaxY - duMinY,
   x1, y1, x2, y2 );

// Read the data from the terrain database
TSLTerrainDataItem *dataItems =
  new TSLTerrainDataItem[ m_terrainGridWidth * m_terrainGridHeight ];
if( m_terrainDB.queryArea( x1, y1, x2, y2, m_terrainGridWidth,
                           m_terrainGridHeight,
                           dataItems ) != TSLTerrain_OK )
{
  return false;
}

// Convert the terrain database to contour vertices so we can give
// them to the contour object
TSLTerrainContourVertexList *vertices =
                                    new TSLTerrainContourVertexList();

for( int i = 0; i < m_terrainGridHeight; ++i )
{
  for( int j = 0; j < m_terrainGridWidth; j++ )
  {
    vertices->addVertex(dataItems[(i * m_terrainGridWidth ) + j].m_x,
                        dataItems[(i * m_terrainGridWidth ) + j].m_y,
                        1,
                        &dataItems[(i * m_terrainGridWidth) + j].m_z);
  }
}

// Height information is now stored in the vertex list so the data
// from the terrain database is no longer required
delete[] dataItems;

TSLTerrainContour contour = new TSLTerrainContour();

// Give our vertex list to the contour object so we can then perform
// contouring - the contour object assumes ownership of the vertex
// list
contour->setVertices( vertices );
```

Although the contour object assumes ownership of the vertex list, the data contained within the list can still be modified by the application without having to generate a new vertex list and setting it on the contour object. This avoids having to do large copies when you wish to modify the data used for contouring. If this is done, the `TSLTerrainContour` object should be informed of the change via the `notifyChanged()` method in order to ensure that the updated data is used for future contouring operations.

C.2.2 Types of Contours

Contour information can be generated either as polygons or lines. When generating contours as lines there are three different algorithms that can be used, specified by the `TSLTerrainContourLineType` enumeration. The simplest of these is `TSLTerrainContourLineTypeSimple` which uses a Triangulated Irregular Network (TIN) to calculate the contour lines. `TSLTerrainContourLineTypeStandard` uses a similar method but performs some optimisation on the resulting contour lines to remove duplicate points from the calculated contours. `TSLTerrainContourLineTypeCONREC` uses a different algorithm that in most cases produces contour lines as good as those generated by the simple or standard methods but is substantially faster.

When generating contours as polygons there is no algorithm choice to make.

C.2.3 Drawing the Contours

Contours generated from the `TSLTerrainContour` class are passed to the application via one of the `TSLTerrainContourCallbacks` virtual methods. Which callback is invoked is dependent on which type of contour (see section 17.8.2) was requested according to the following table:

Callback	Used by
`TSLTerrainContourCallbacks::progress`	All

`TSLTerrainContourCallbacks::drawLine`	`TSLTerrainContour::drawContourLine` using the following types:
	`TSLTerrainContourLineTypeSimple`
	`TSLTerrainContourLineTypeCONREC`

`TSLTerrainContourCallbacks::drawPolygon`	`TSLTerrainContour::drawContourPolygon`
`TSLTerrainContourCallbacks::drawPolyline`	`TSLTerrainContour::drawContourLine` using the following types: `TSLTerrainContourLineTypeStandard`
`TSLTerrainContourCallbacks::drawText`	`TSLTerrainContour::drawContourLine` using the following types: `TSLTerrainContourLineTypeStandard`
`TSLTerrainContourCallbacks::drawTIN`	`TSLTerrainContour::drawTIN`

You should override each of the callbacks that will be used for your selected method of contour generation. The `TSLTerrainContourCallbacks` class provides default implementations of all the callbacks so that you only need to implement the ones that you are interested in.

The callbacks will be invoked numerous times before the original draw call returns. In order to prevent excessive redrawing your application should wait until the draw call has returned before updating the display of your application.

This example shows an implementation of the

`TSLTerrainContourCallbacks::drawPolyline()` callback in which the generated contour lines are added to a `TSLStandardDataLayer` to be drawn to the screen after contour generation has finished.

```
void TerrainContouringView::drawPolyline (TSLTerrainContourVertexList*
vertices, double attribute)
{
  // The coordinates of the vertices given to us are in the coordinate
  // system of the terrain data, which may not be the same as that of
  // the map we have loaded. Therefore it may be necessary to
  // convert the coordinates so the contour lines appear in the correct
  // place on the map.
  TSLCoordinateSystem *terrainCS =
                              m_terrainDatabase->queryCoordinateSystem();
  const TSLCoordinateSystem *mapCS =
                                 m_mapDataLayer->queryCoordinateSystem();
  bool needToConvert = false;
  if( terrainCS->id() != mapCS->id() ||
      terrainCS->getTMCperMU() != mapCS->getTMCperMU() )
    needToConvert = true;

  TSLCoordSet *coords = new TSLCoordSet();

  // Process the list of vertices given to us into a polyline so we can
  // display it on the map in a standard data layer
  for( int i = 0; i < vertices->numberOfVertices(); ++i )
  {
    TSLTerrainContourVertex &vertex = vertices->at(i);
    TSLTMC tmcX = 0, tmcY = 0;

    if( !needToConvert )
    {
      // The terrain database and map coordinate systems are the same
      terrainCS->MUToTMC( vertex.x(), vertex.y(), &tmcX, &tmcY );
    }
    else
    {
      // Convert between the terrain database and map coordinate systems
      double lat = 0.0, lon = 0.0;
      terrainCS->MUToLatLong( vertex.x(), vertex.y(), &lat, &lon );
      mapCS->latLongToTMC( lat, lon, &tmcX, &tmcY );
    }

    coords->add( tmcX, tmcY );
  }

  TSLEntitySet *es = m_contourLayer->entitySet();
  TSLPolyline *line = es->createPolyline( 0, coords, true );
  if( line )
  {
    line->setRendering( TSLRenderingAttributeEdgeStyle, 1 ) ;

    // Determine line colour based on the height of the contour
    long colour = ( 255 / m_maxTerrainHeight ) * attribute;
    line->setRendering( TSLRenderingAttributeEdgeColour,
    TSLDrawingSurface::getIDOfNearestColour( colour, 0, 255 - colour) );
    line->setRendering( TSLRenderingAttributeEdgeThickness, 1 ) ;
  }
}
```

C.2.4 Drawing the Contour Labels

When drawing contour lines using `TSLTerrainContourLineTypeStandard` there is the option to draw labels for the generated contour lines. This is enabled by passing a non-NULL value to the `textPrefix` parameter of the `TSLTerrainContour::drawContourLine()` method, which will be passed to the `TSLTerrainContourCallbacks::drawText()` callback. This is usually set to a description of what the value in the label will represent (e.g. 'Height:' or 'Temperature:'), but if nothing is desired can be set to an empty string.

When using text labels with the alignment value set to `TSLVerticalAlignmentMiddle` the contour lines are split at appropriate points around the labels so that the lines do not run through the labels themselves. As the contents of the labels are controlled via the application by the `TSLTerrainContourCallbacks::drawText()`, this necessitates informing the contour object of the maximum length that the text strings will be when the `TSLTerrainContour::drawContourLine()` method is invoked.

One way of doing this is to create a dummy text object of the longest expected length and use this to determine the size to pass in as follows:

```
TSLText *textObj = m_contourLayer->entitySet()->createText( 0, 0, 0,
                                    maxLengthLabel.str().c_str(), 100 );

// It is necessary to set up the following attributes on the text object
// for updateEntityExtent() to work
textObj->setRendering( TSLRenderingAttributeTextSizeFactor,
                       m_textSizeFactor );
textObj->setRendering( TSLRenderingAttributeTextSizeFactorUnits,
                       TSLDimensionUnitsMapUnits );
textObj->setRendering( TSLRenderingAttributeTextFont, 2 );

// Set an entity ID on the temporary text object so we can remove it
// once we're done
textObj->entityID( INT_MAX );

m_contourLayer->notifyChanged();

// Store the currently viewed area of the map. In order to calculate the
// extent of the text object we need to change the viewed area so that
// our temporary text object would be visible
double viewedUUX1, viewedUUY1, viewedUUX2, viewedUUY2;
m_drawingSurface->getUUExtent( &viewedUUX1, &viewedUUY1,
                               &viewedUUX2, &viewedUUY2 );

double newUUX1, newUUY1, newUUX2, newUUY2;
long newSizeArea = 2 * m_textSizeFactor;
m_drawingSurface->MUToUU( -newSizeArea, -newSizeArea,
                          &newUUX1, &newUUY1 );
m_drawingSurface->MUToUU( newSizeArea, newSizeArea,
                          &newUUX2, &newUUY2 );
m_drawingSurface->resize( newUUX1, newUUY1,
                          newUUX2, newUUY2, false, true );

// Now calculate the size of our text object
m_drawingSurface->updateEntityExtent( textObj );
TSLEnvelope env = textObj->envelope( m_drawingSurface->id() );
unsigned long envWidth = env.width();

// Now we have the width of the text object in TMCs we need to convert this to
the terrain database units
double lat1, lon1, lat2, lon2, x1, y1, x2, y2;
m_drawingSurface->TMCToLatLong( env.bottomLeft().x(),
                                env.bottomLeft().y(), &lat1, &lon1 );
m_drawingSurface->TMCToLatLong( env.topRight().x(),
                                 env.topRight().y(), &lat2, &lon2 );
m_terrainDB.latLongToMU( lat1, lon1, &x1, &y1 );
m_terrainDB.latLongToMU( lat2, lon2, &x2, &y2 );

// This is the width of the text labels in the terrain database units
// with some additional space either side
width = ( x2 - x1 ) * 1.5;

// Now we have the width we no longer need our text object
m_contourLayer->removeEntity( INT_MAX );

// Finally, reset the viewied area of the map back to what it was originally
m_drawingSurface->resize( viewedUUX1, viewedUUY1,
                          viewedUUX2, viewedUUY2, false, false );
```

C.2.5 Performance Notes

Calculating contours can take a considerable amount of time when given large amounts of data to work on. As the result of a draw operation will not change if the data remains the same, it is more sensible to store the results of the contouring operation in a form that

allows for fast rendering. The example in section 17.8.3 does this by creating geometry objects for each contouring line and storing them in a `TSLStandardDataLayer`. This prevents needless recalculation of the same points on each draw in the application.