SpiderFusion

KONGSBERG

FEATURES

- SpiderFusion is a modular SW toolset
- Created for easy and efficient implementation of end-user software solutions
- Support for processing information in multi-threaded environments
- Functionality for extending the Qt SW toolset
- Management and control of Application Objects (AO)
- Complex data structures connected to the geographical space, including related computation functions
- Interaction between software and its platform, i.e. hardware, programming language and operating system

SpiderFusion

SpiderFusion is a software development toolset providing easy and efficient implementation of end-user software solutions. Using this type of toolset is a common software development approach, and hence, there are numerous toolset products on the market for solving different type of problems.

During the last 15-20 years, more and more emphasis has been put on basing toolset implementation on the principles, structures and prescriptions defined by modern international standards. This approach will guarantee that the toolset solves their tasks according to widely recognized principles. This will also simplify toolset replacement.

An alternative approach is to skip the toolset level, and instead, implement the entire software directly from the prescriptions defined by the standard. This approach will certainly become more expensive, but could also lead to exotic solutions and reduced quality. Standardized toolsets will also provide more smoothly interactions within and between software components.

The structure of SpiderFusion is modular. It basically consists of 5 modules. Furthermore, each module is divided into a varying number of components of different size.

Note in particular that the 4 uppermost modules do not depend on each other. This means that SpiderFusion could be installed in various configurations being combination of one or more modules. For instance, many projects will only use the SpiderSpatial module. Furthermore, we see that all 4 modules depend on the SpiderGear module. This means that all installation of SpiderFusion must include the SpiderGear module. SpiderGear could also be used stand-alone.
The component level does not allow individual components to be installed, even if this would have been technically feasible.

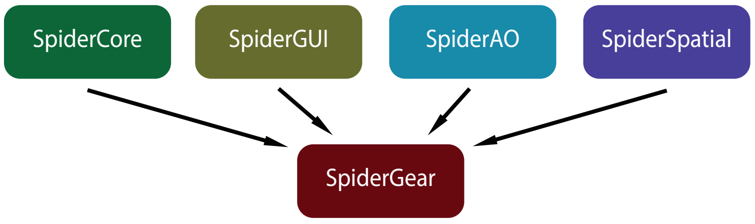


Figure 1: SpiderFusion modules and their interdependence

Below are descriptions of the main focus for each of the SpiderFusion modules. Furthermore, it describes each individual component on an overall level of detail. SpiderFusion User Guide documentation has more detailed technical information on the concepts and functionality of individual components. Detailed technical information is available as Reference Guide information meant for software developers.
information is available as Reference Guide information meant for software developers.

SpiderCore

SpiderCore offers software development support for information to be processed in a multi-threaded runtime environment. This means that separate parts of the program (“threads”) could be executed in parallel and where the threads are given the possibility to share memory.

During the last couple of decades, modern processor technology has been given the ability to support processing of an increasing number of threads in parallel. This possibility has been heavily used by modern software development, even if it increases software complexity and leads to higher risks for separate threads interfering negatively with each other.

SpiderCore provides tools and software patterns for simplifying this problem, and hence minimizing the level of potential conflicts. This approach makes the final program more robust, and also often more efficient.

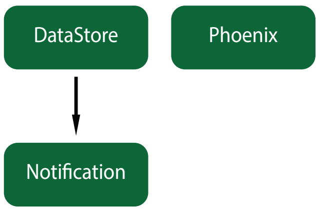


Figure 2: SpiderCore toolset components and their interdependency

The *Notification* component provides tools for safe and robust communication between different threads. This is a necessary basic prerequisite infrastructure for robust multi-thread management.

The *DataStore* component provides tools for managing common information in computer memory shared by several processing threads, by preventing them from interfering with each other.

In order to do so, the common memory must be divided into individual sections corresponding to the elements («data-objects») of the common data structure of the program. This approach allows the programmers to work with the native data types and data structure supported by the programming language without paying any particular attention to sharing this information between parallel threads.

Threads will also need to subscribe for messages (via the Notification component) when the stored value of a particular data element has been modified by another thread.

The *Phoenix* component provides functionality for listening for low-level signals from hardware components or from the operating system, and then transforming them into normal software error messages.

The motivation for this type of transformation is to prevent the program from terminating abruptly. Instead, the program could finish ongoing activities and then terminate in a controlled manner.

SpiderGUI

SpiderGUI provides functionality for extending the software toolset «Qt».

Qt is a very popular commercial toolset for developing for instance platform independent Graphical User Interfaces ("GUI"). A GUI is defined as the part of the software that is "visible" on a screen, and hence is used for communicating information between a program and the operator.

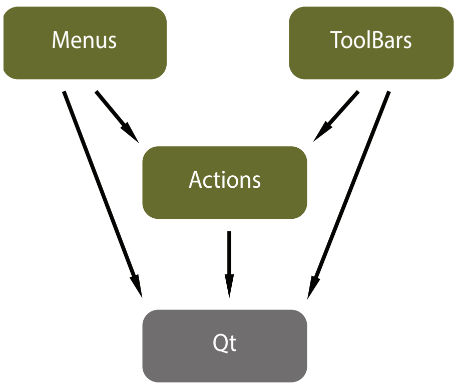


Figure 3: SpiderGUI toolset components and their interdependency

The SpiderGUI extension is mainly focusing on useful tools to simplify the software development of complex systems with complex GUIs. This means that SpiderGUI is often used in combination with components from SpiderCore and SpiderGear.

The *Menus* component provides functionality for efficient development and configuration of the menus of a GUI. It is particularly useful to be able to configure the menus after the program has been built. This feature simplifies maintenance, and provides efficient re-use of menu elements.

The *ToolBars* component provides functionality for efficient development and configuration of the toolbars of a GUI. It is particularly useful to be able to configure the toolbars after the program has been built. This feature simplifies maintenance, and provides efficient re-use of toolbar elements.

The *Actions* component provides tools for management of pre-built functions to be executed in large and complex systems. One particular "Action" element could be connected to one or more menu toolbar elements.

SpiderAO

SpiderAO provides powerful tools for management and control of a software science concept called «Application Objects» («AO»). These are the result of splitting a runtime software program into formal runtime components («AO components»). This type of organization is very useful in complex software systems.

Using SpiderAO for this purpose simplifies the configuration of such type of systems and offers simple re-use between sub-systems and between projects. The Framework component provides the framework needed for developing and executing AO-components.

The Configurator component provides functionality for simple and flexible configuration of AO components, which in turn simplifies the process of building, maintaining and extending large software systems.

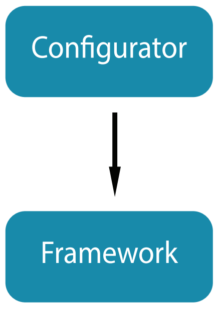


Figure 4: SpiderAO toolset components and their interdependency

SpiderSpatial

SpiderSpatial provides various general data structures connected to the geographic space, such as *coordinates, coordinate systems geometries, networks and geographic fields*. Furthermore, these structures provide many general computation functions such as *coordinate transformations, triangulation and logical operators*.

These are functions with a mathematical complexity beyond the skills of an average programmer. Hence, by using SpiderSpatial, you could represent your own structures and perform complex computations by writing less code without this skill. The code would also be thoroughly tested and significantly more robust.

SpiderSpatial has been designed according to modern software development principles. It is heavily influenced by the basic concepts and data models defined by the ISO-19100 family of formal international civil standards for geographic information. These also play the role as base standards for similar civilian European (CEN) and Norwegian (SOSI) national standards. SpiderSpatial also uses other ISO standards on the more detailed level. On the technical side it takes advantage of the large family of technical and more non-formal standards from OGC (Open Geospatial Consortium). Finally, SpiderSpatial supports a small number of open military format standards for information interchange, such as STANAG-3809 (DTED) standard for terrain elevation data also used on the civilian side.

SpiderSpatial is particularly useful as supplement of geospatial structures and functionality to graphical display systems. Together, they may form a basic infrastructure for map display systems.

SpiderSpatial is also often used as a support library for geospatial computation functions, such as line-of-sight or navigation functions.

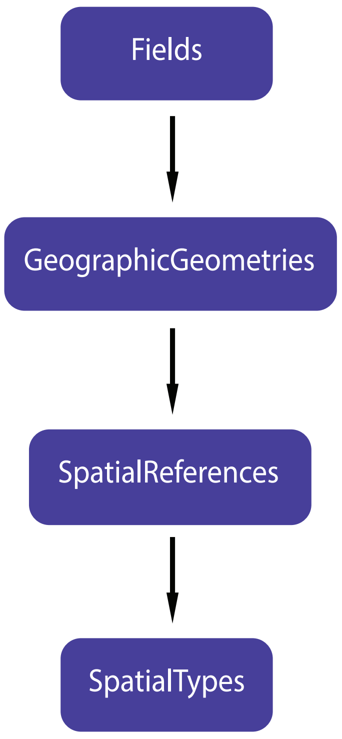


Figure 5: SpiderSpatial toolset components and their interdependency

SpatialTypes defines a large number of data structures for representation of geospatial information within the context of the programming language C++. This means that it provides many of the basic structural building blocks needed for geospatial information. It puts a lot of emphasis on data type safety in order to protect the programmer against making trivial errors.

SpatialReferences provides representation of the most important spatial reference systems and coordinate systems used by geospatial information. The most important benefit is the large number of computational functions available for coordinate conversions between reference systems, and for geometrical constructions. These functions constitute the basic building blocks needed for processing of geospatial information. It puts a lot of emphasis on hiding most of the complex mathematics behind the computation functions.

GeographicGeometries defines a large number for formal geometries used for geo-referencing real-world objects from the application domain. Geometries may also have topological properties, such as network properties representing road network and support navigation within road networks. Geometries also provide useful analysis functions, for instance for computing overlaps between two or more geometries. The representation of the geometries is very accurate on the ellipsoid-shape globe, and also provides conversions of geometries to other spatial reference systems.

Fields defines various data structures for representing values varying over the geographic space, such as precipitation or altitude. The main benefit is representing complex information without any performance penalty. Fields have traditionally been the preferred analysis approach for geographic information by allowing different field types interact spatially with each other.

SpiderGear

SpiderGear provides basic functionality for the interaction between own software and its “platform” (i.e. hardware, programming language and operating system). The interaction could then be made independent of the underlying platform type. This means that the software could be moved smoothly between different platform types and platform versions without any modification.

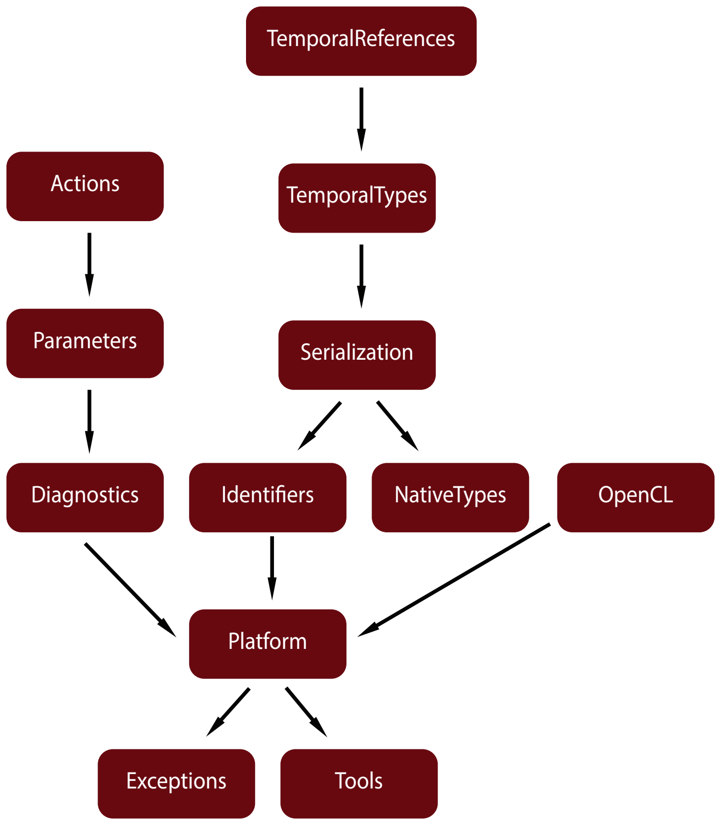


Figure 6: SpiderGear toolset components and their interdependency

The tools available in this module are used by other modules in SpiderFusion and other components of SpiderGear, as well as by software using SpiderFusion.

The *Exceptions* component provides functionality for managing error situations during runtime. The component utilizes the powerful “exceptions” concept available in modern programming languages. The component provides tailoring of own software, as well as help for implementing efficient error handling and error tracing.

The *Tools* component provides a collection of various tools being too small for being organized as separate components. The tools are mainly useful as “everyday tools” for the programmer. Most of them could be considered as useful extensions of the programming language making the programming work easier and more robust.

The *Platform* component provides functionality simplifying the interaction with the hardware or the operating system. This is particularly useful if you plan to run the software on different operating system types such as Windows and Linux. Hence, the component provides a neutral interface, preventing us from changing our own software during the porting from one operating system to another.

The *Diagnostics* component provides software performing runtime status diagnosis. This is primarily meant as an efficient tool to be used during the software development phase, but it could also be used for discovering error during normal software execution.

NativeTypes is a small component providing platform independent representation of the programming language native data types.

Identifiers is a small component providing generation of standardized identifiers for software objects. An identifier is defined as a unique code (analogue to a social security number) representing this particular software object. Identifiers are often used in modern programming, and in particular in connection with distributed systems. The various identifier types are defined by international standards.

The *OpenCL* component has its name from an open source standardized technology for programming of modern processors designed for massive parallel processing, such as graphic processors. These processors could be used to obtain extreme computing performance in dedicated situations. The OpenCL component provides software tools simplifying programming of this technology in a platform independent way.

The *Parameters* component provides tools for simplifying the programming of some basic and often used software patterns.

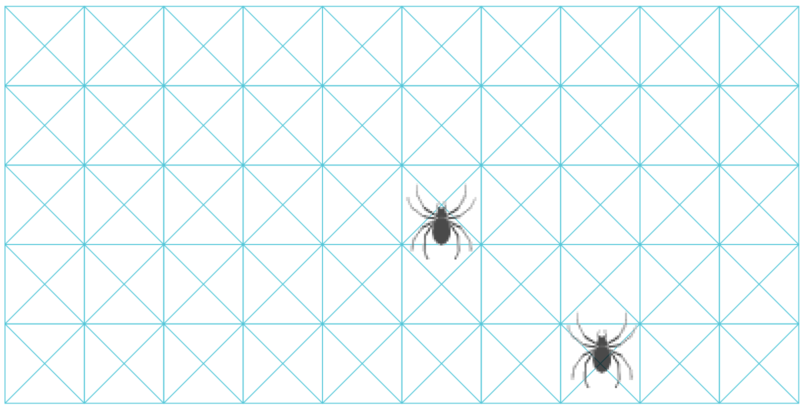
The *Serialization* component provides tools for simplifying the process of storing and restoring runtime data to and from devices for permanent storage.

The *Actions* component provides tools for organizing of ready-to-use functions to be executed in large and complex software environments.

The *TemporalTypes, TemporalReferences* and *TemporalGeometriers components* provide functionality for representing and processing "time". An example is the representation of "time" in different reference systems such as clocks, time zones and calendars. In addition, it provides functionality for comparison and conversions of point-in-time and periods.

References:

ISO: http://www.iso.org
CEN: http://www.cen.eu
SOSI: http://www.statkart.no
OGC: http://www.opengeospatial.org
ESRI: http://www.esri.com



Contact person:

Kari Anne Sandengen
kari.anne.sandengen@kongsberg.com