flight software development kit

mission ready software.

datasheet

Flight Software Development Kit (FSDK) is a unique, innovative development environment, which permits the creation of mission-specific spacecraft flight software using configurable, off-the-shelf software components. It comprises three major parts: an extensive library of validated components, a lightweight framework which promotes portability and modularity, and tooling to support component development, software integration and test.

extensive component library

The FSDK component library includes a wide range of components covering all mission needs. As with all elements of our FSDK, components are developed to a strict coding standard based on industry norms for mission-critical software.

Subsystem components: for many commercial off-the-shelf products such as electronic power systems, attitude control systems and radios

Data handling components: gathering, pooling, logging and reporting telemetry parameters from the complete system

Monitoring components: checks on parameters to validate correct system operations

Communications components: monitoring, control and reporting to and from ground or other systems using a variety of standard protocols

Automation components: automation of onboard activities such as responding to events and scheduling onboard operations based on relative time, absolute time or spacecraft orbit

Mission components: manage the spacecraft mode and separation sequence

development tooling

FSDK tooling is built on a variety of standard and cross-platform technologies and can be readily adapted to suit different processes. Graphical tooling integrates with the Eclipse environment, whilst the complete build system is fully accessible from the command line and can be easily automated. FSDK tooling helps guide engineers through the software development process rapidly, promoting an agile and iterative approach.

test support tooling

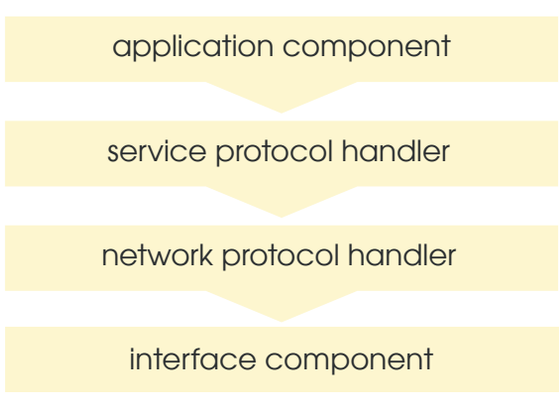
Support for development and integration testing is provided in the form of graphical and command line tools which can communicate with flight software, acting as 'ground'. The graphical TMTC Lab operates using the software model and permits all space-ground interactions to be tested with ease. Additional libraries, including support for Java and Python, allow sophisticated automated test scripts to be developed using the power of the software model.

modular communications

The FSDK is designed to construct modular communications stacks from components. Interfaces between components follow a standard pattern based on services.

This approach allows individual layers of the stack to be modified or substituted without impacting layers above and below. By replacing the lowest communications layers, interfaces or links can be emulated for testing.

As protocols are kept separate from the functions of the software, the FSDK allows different protocols to be used without impacting software capabilities.



bright ascension

FIND OUT MORE

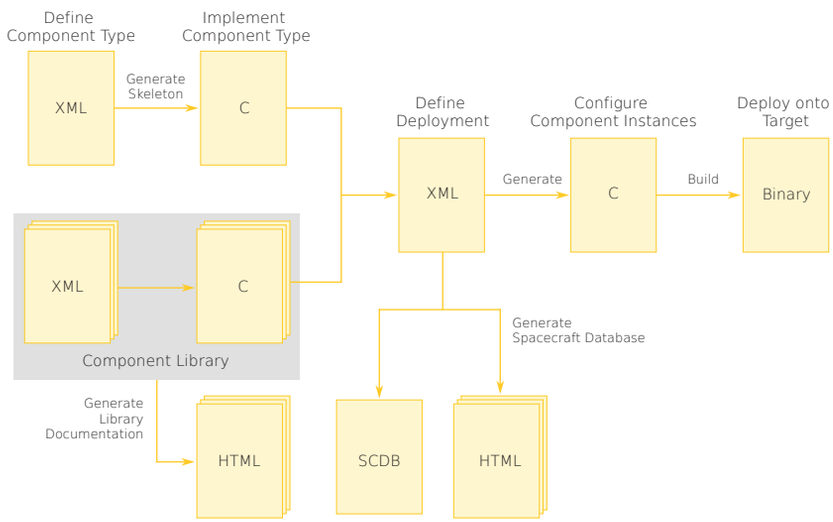
+44 (0) 1382 602041
enquiries@brightascension.com
www.brightascension.com

FOLLOW US

@brightascension
@bright-ascension-ltd

development tooling workflow

The core of the FSDK is the model, which captures the architecture of the software being built in terms of the components being used. The interfaces to all types of components are described in a short XML file together with documentation. These component types can then be instantiated and assembled together to make a fully-functional flight software deployment according to a flexible specification. The tooling uses this specification to generate simple and efficient code, specific to the flight software deployment, which 'glues' the components together and lets them interact. Code generation is also provided to speed up component development and provide templates for component unit testing using the Unity and CMock tools.



Using information from the model, additional tooling is able to generate full documentation, for developers or operators, which describes the components and how they can be used. The GenerationOne model can also be exported, allowing it to be used by external applications such as the GenerationOne Mission Control Software.

platform abstraction and services

Services allow components to interact in standard ways and are at the heart of the GenerationOne component model and framework. The FSDK framework is built around a lightweight service mechanism which links components using the information from the underlying GenerationOne model.

Defining component interfaces in terms of services gives many advantages:

- Key functions can be separated from communications protocols allowing much more efficient use onboard and enabling greater software reuse
- Standard services for important activities, such as working with telemetry parameters, permit operators to work at a higher semantic level
- Applying standard services to input/output onboard abstracts components away from the underlying platform and promotes portability
- Location-independent services allow the easy distribution of flight software across multiple computing platforms without adding operational complexity

rapid and portable development

The extensive component library of the FSDK, together with the development tooling, permits the rapid development of mission-specific flight software. By making use of the portability and modularity features of the FSDK, your flight software can be targeted to execute on a desktop workstation, allowing early prototyping, debugging and team development. Similarly, it is trivial to replace subsystem software components with simulations, permitting development and testing even if key hardware is unavailable.

documentation and tutorials

The FSDK includes full source code for all flight software and is accompanied by an in-depth User Manual. Also included are a number of tutorials and fully-featured examples, carefully designed to introduce all key FSDK concepts and to allow the rapid development of mission-specific flight software.

platform and subsystem support

Through our unique, efficient approach to platform abstraction, we can also support a range of operating systems from vanilla Linux to hard real-time environments such as FreeRTOS and RTEMS. Library components allow access to system peripherals and subsystems in a regular way, permitting a high degree of modularity. Where the underlying platform provides fully featured drivers, such as in a Linux environment, driver components are simply responsible for making these accessible to the rest of the system. In other environments, especially when running on bare metal, drivers are written as components making them fully portable.

bright ascension

FIND OUT MORE

+44 (0) 1382 602041
enquiries@brightascension.com
www.brightascension.com

FOLLOW US

@brightascension
@bright-ascension-ltd