<u>F</u>ile <u>E</u>dit <u>N</u>avigate Search <u>P</u>roject <u>T</u>ools <u>W</u>indow <u>H</u>elp 🗄 Report Navigator 🖻 🕀 🔄 🕐 ETC_Call_Pair_06 × 🛃 engine.c:79 🗙 😑 update_hydraulic_ 🔺 DCCC for Coupling: ETC_Call_Pair_06 -Code:engine.c:79 calc_friction update_pistor Find element 🗸 🔎 🗘 😓 🛛 🤊 Instrumented Source 😑 update_drivetrain request_power ▼ Coupling Metadata for(int i = 0; i < 2; i++){
 if((get_max_power(max_power));
}</pre> 393 get_grip_level 394 # 69 "src/engine.c" 3 4 • Calling function: update_tc e_stop_check 395 396 397 get_TC_mode • Called function: set_max_power e_stop_check Interface: engine-TC # 69 "src/engine.c' 398 399 400 update wheels Name: ETC_Call_Pair_06 }else{ get max power engine.driven_wheels # 71 "src/engine.c" 3 4 Filter: All 👻 Tables: 🐻 📄 401 request_power Couplings 402 get_max_power 403 404 405 406 407 #Couplings #Satisfied Satisfied % #Tests Name # 71 "src/engine.c" set_max_power update_tc × ETC Call Pair 06 🖳 🗐 Couplings 🖻 🌄 engine-TC 408 }else{ Coupling Transitions 409 410 🕀 🎯 Control Couplings for inte //get_min for driven wheels } ETC Call Pair 01 Seen Name Location From To }} 411 ETC Call Pair 02 412 413 414 site in function update_tc 0 1 × ETC_Call_Pair_03 void set_max_power(enum T_STATUS cur switch(current_status){ case STOP: in function set_max_power 1 2 target ETC_Call_Pair_04 415 ETC_Call_Pair_05 - 2 🛏 🗙 target 416 417 0 🛏 🗙 site - 1 max_power = 0; break; × ETC_Call_Pair_06 × ETC_Call_Pair_07 case SAFETY: 418 419 420 421 max_power = 20; break; 🖃 爹 Global Data Couplings for ETC_Global_Data_01 Name Parent Location case LIMITED: ETC_Global Data 02 max_power = 40; break; case FULL: ETC_Call_Pair_06 : ETC_Call_Pair_06 site in function update_tc 422 422 423 424 425 ETC_Global_Data_03 ETC_Call_Pair_06 : ETC_Call_Pair_06 target in function set_max_po ETC Global Data 04 max_power = 100; break; ETC Global Data 05 426 427 ETC Global Data 06 } Coupling Results Filter: All 🔻 ETC_Global_Data_07 428 }} 429 🖶 🌉 mathsLib-suspension No data available. 430 431 int get_max_power(enum T_STATUS curr 🕞 🎯 Control Couplings for switch(current_status){ 432 case STOP: 433 434 return 0; case SAFETY: return 20; case LIMITED: return 40; case FULL: 435 436 437 438 439 440 441 return 100; } }} 442 443 void request_power(int reque <



Report

Safety through quality

PRODUCT PREVIEW

RapiCoupling

Data Coupling and Control Coupling Analysis with Rapi**Coupling**

Source

Product preview: RapiCoupling

RapiCoupling

How can RapiCoupling help you?

Rapi**Coupling** provides an automated framework to support Data Coupling and Control Coupling (DCCC) analysis for DO-178C and ISO 26262. It supports a wide range of coupling types and allows flexible definition of analysis goals.

Benefits of using Rapi**Coupling**

Rapi**Coupling** helps support the development of reliable software by providing a flexible, automated approach to data coupling and control coupling analysis. By using Rapi**Coupling**, you can:

- Automate instrumentation and observation to provide DCCC analysis evidence using a qualified workflow.
- Configure DCCC analysis to meet your project's needs.
- Seamlessly integrate DCCC analysis into your existing development environment.
- Trace between tests, code and DCCC goals.
- Reduce your reporting effort by merging DCCC coverage from different runs and builds, even when they have different instrumentation.

Rapi**Coupling** use cases

- DCCC analysis to meet DO-178B/C objectives.
- DCCC analysis to contribute to AC 20-193 / AMC 20-193 objective MCP_Software_2.
- Data flow and control flow analysis to meet ISO 26262 guidelines.
- Generate evidence that interactions between software components have been exercised sufficiently in testing.
- Provide assurance that software interface requirements are met by your code.
- Diagnose missing DCCC coverage to improve the quality of your requirements, tests and code.

How does RapiCoupling work?

Rapi**Coupling** provides a powerful framework to help you define couplings automatically, refine them manually, and automatically collect and report results.



This involves the following process:

- 1. First, the software architecture is defined by identifying the components and interfaces between them (as implied by DO-178C objective 6.4.4.d). Fast automated extraction of the architecture is available when the code structure permits.
- Next, couplings (coverage and other DCCC goals) are created. Couplings can be generated automatically from the code and architecture, or added manually/ programmatically. Rapi**Coupling**'s flexible coupling definition approach allows you express a rich variety of properties you wish to observe in testing.
- 3. Rapi**Coupling** applies instrumentation to the code, so when tests are run, results are collected and a report is generated.
- Summary and detailed report views help you identify and diagnose missing couplings, and exports support using Rapi**Coupling** to provide certification evidence.



This is a "Product Preview" with preliminary information about a product on our roadmap. To ensure that we can deliver the best product to meet your needs, we invite your feedback on requirements, design and evaluation – contact us to be involved.

Key features

The features listed below are features expected for the first public version of Rapi**Coupling**. We have plans to develop the product further when the first version has been released. For information on our long-term roadmap and to get involved, contact Rapita at info@rapitasystems.com.

DCCC analysis

- Automated instrumentation and observation for software Data Coupling and Control Coupling
- Analyze Data Coupling and Control Coupling structural coverage
- Analyze interface constraints derived from your architecture
 - Check that constraints are observed and not violated
 - e.g. data constraints (range, relationship), call sequencing constraints
- Generate common couplings automatically by static analysis
 - Call-return couplings
 - Definition-use data couplings (parameters)
 - Definition-use data couplings (global variables)
- · Create couplings manually
- Configurable analysis to create DCCC criteria that are right for your project
- Architecture-led coupling definition
 - Define components to match your system architecture with automation assistance (files, folders, functions)
 - Create interfaces to capture the nature of interactions, with automation assistance
 - Generate couplings from interfaces between components by static analysis, link manually created couplings to interfaces
- Visualize source code architecture and dependencies
- Traceability
 - Identify relationships between source code and architecture
 - · Add links to trace artifacts to couplings
- See how each test run or set of tests contributed to observed results
- Diagnose missing coverage
- Merge results from different integration builds and test runs

Language support

- C, compilers including Visual Studio®, GCCTM, Diab® and TASKING®

Build integration

- Multiple strategies available:
 - Compiler wrappers
 - Clone integration
 - Scripting into build system directly
- Support for very large code bases
- Split instrumentation between builds

Target integration

- Support for data collection using CAN, Serial, Ethernet, debuggers and our **RTB**x data logger
- Low overhead data collection
- No dynamic memory requirements
- · Collect and report results on a per-test basis
- Analysis across power cycles (subject to hardware requirements)

Integrated testing environment

- Summary and detailed results views
- Filter results on interface, test case, & coupling observed, unobserved or violated
- Coupling Navigator to easily navigate couplings
- Code viewer:
 - View original source code, pre-processed and instrumented code
- Compare reports
- · Database-like search function
- Multi-user testing environment

Compatibility

- Runs on host operating systems
 - Windows[®] 10+ and Windows Server[®] 2016+
 - Linux[®] distributions including Ubuntu[®]
- Results can be collected from systems without supported operating systems and transferred to a supported system for analysis

Licensing

- Enterprise license gives you access to new versions, support and maintenance
- One-year support and maintenance included in purchase price
- Single price for all features
- Licenses transferable across projects





About Rapita

Rapita Systems provides on-target software verification tools and services globally to the embedded aerospace and automotive electronics industries.

Our solutions help to increase software quality, deliver evidence to meet safety and certification objectives and reduce costs.

Find out more

A range of free high-quality materials are available at: rapitasystems.com/downloads

SUPPORTING CUSTOMERS WITH:

Tools	Engineering Services	Multicore verification
Rapita Verification Suite:	V&V Services	MACH ¹⁷⁸
Rapi Test	Integration Services	Multicore Timing Solution
Rapi Cover	Qualification	
Rapi Time	SW/HW Engineering	
Rapi Task	Compiler Verification	
Rapita Verification Suite : Rapi Test Rapi Cover Rapi Time Rapi Task	V&V Services Integration Services Qualification SW/HW Engineering Compiler Verification	MACH ¹⁷⁸ Multicore Timing Solution

Contact

Rapita Systems Ltd. Atlas House York, YO10 3JB UK

+44 (0)1904 413945

Rapita Systems, Inc. 41131 Vincenti Ct. Novi, Mi, 48375 USA +1 248-957-9801

Rapita Systems S.L.

Parc UPC, Edificio K2M c/ Jordi Girona, 1-3 Barcelona 08034 Spain +**34 93 351 02 05**





linkedin.com/company/rapita-systems



info@rapitasystems.com