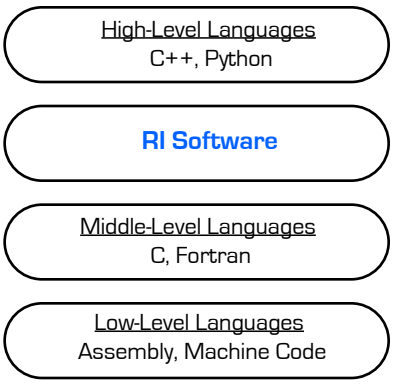
RI Software: The State-Based Approach

Cassini was designed from the ground up to have a more comprehensive interaction of hardware and software. This is achieved by using a state-based approach in the software, allowing measurement functions and commands to operate on the physical resources within the tester at a hardware-state level. What this means for a user on Cassini is that the programming interface and workflow reflect this state-based method when it comes to developing measurement algorithms and building test plans. The following section explains this software and programming approach, how it is used to build measurements, create test plans, and control the execution flow.

What is the state-based approach?

Programming languages can be classified and grouped based on their abstraction level. The lowest level of abstraction is machine language, combinations of binary 1's and 0's that are driving control hardware at the transistor level. All languages above machine language use syntaxes and expressions to represent these groups of 1's and 0's in order to be understood by a user and facilitate program design. These commands are then translated back to hardware specific binary commands by a compiler. The higher the abstraction level, the simpler the user commands for complex tasks become, making expansive and intensive computational tasks manageable for a programmer. The tradeoff of higher level languages is that compiled code loses efficiency when executed in the hardware. State-based programming is an approach that offers both ease of use for programming and higher code efficiency in hardware execution. This is accomplished by coupling the software function's execution to changes in hardware states. For example, to sweep the frequency in a signal generator, the device transitions from one frequency to the next by incrementally shifting the frequency "state" and holding all other hardware states constant. A conventional programming approach requires that all of the hardware states be explicitly defined in software either by the user or from a high-order command that contains several procedures. This can lead to redundant and/or excessive code executed on the hardware to control the signal generator. Using the state-based approach in Cassini, the software is aware of the current hardware state, and the command to sweep frequency is simplified to

Programming Abstraction Levels

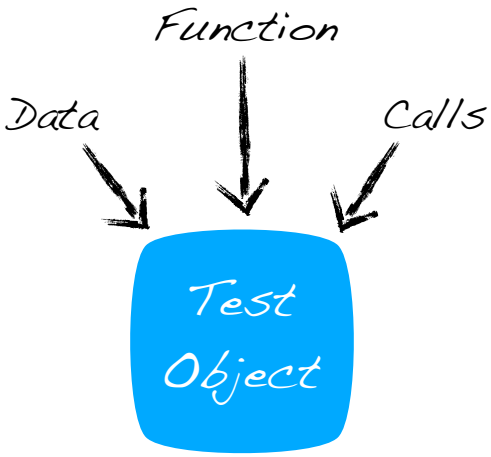


code that only manipulates the frequency states to accomplish the task. This not only reduces the executed commands to the simplest form for the hardware, but provides a finer degree of control that a user can exert in software.

Graphical Programming

Regardless of the language or interface, every program needs a system of logical organization and structure to make development easier. A typical program starts by linking reference libraries that contain functions and macros a user's program will access. Then global default values and variables are defined that will be used to control and store data. Finally, commands and functions are declared in the order they are to be executed. The content of the RI software is no different, but this structure has been reengineered to fit within the context of a graphical programming environment that provides some unique advantages.

To make the interface more intuitive and help the user take advantage of this type of control, the programming interface was designed to mimic state-based logic flow. Commands and functions are represented as blocks in a schematic layout. These basic building blocks, called test objects, encapsulate three essential elements:

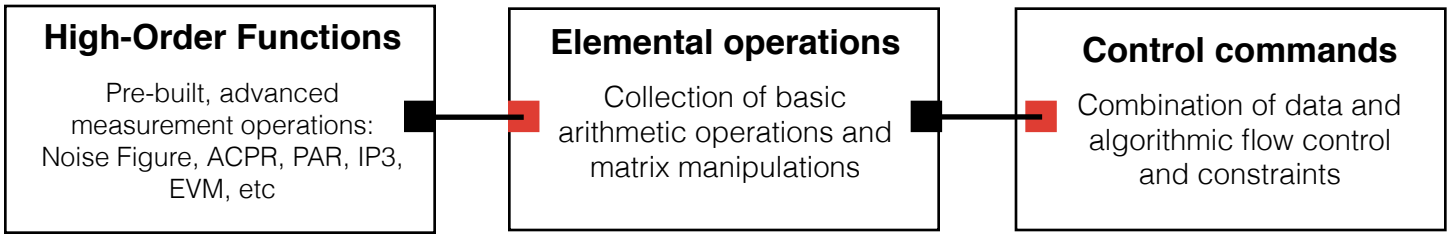


***Data*:** information created or captured
***Function*:** the action or task that acts on the data
***Calls*:** requests generated by the software to utilize resources in either hardware, software, or both

Operations on data are controlled by connecting these block's input/output terminals to each other to build algorithms or measurement functions. By visually connecting actions and functions in this way, the user is provided with a more useful depiction of what the program is doing and a more intuitive way to design and convey complex operations.

Dynamic Test Objects

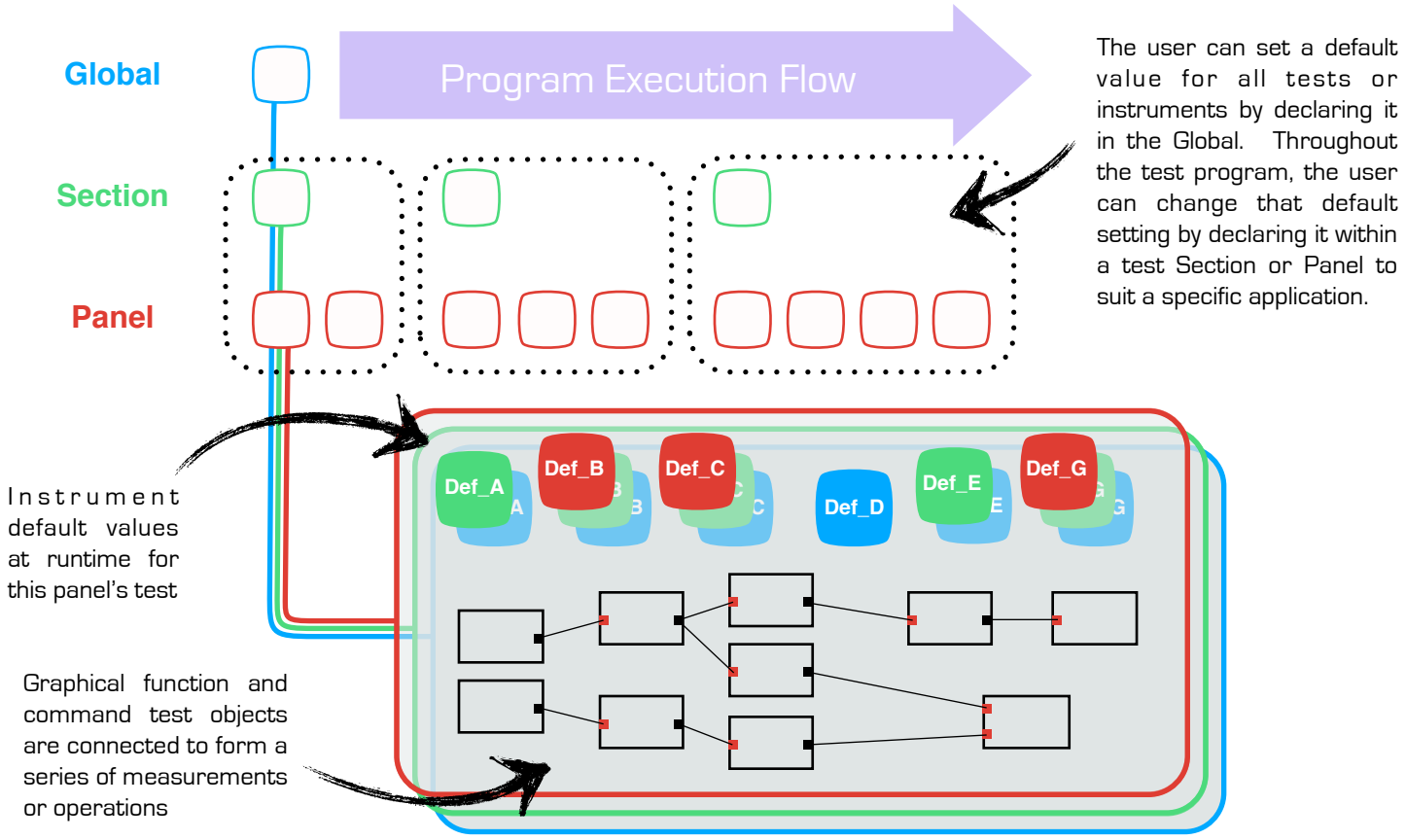
The object calls in each function block link to the hardware resources in Test Instrument Modules connected to Cassini(see "Software Architecture" figure in Philosophy document). A library of function blocks are dynamically assembled to reflect all of the operational capabilities available to the user based on the TIMs attached to the system. Building upon the software's strong correlation to the hardware, the RI test plan software is able to provide three classes of functions in the library that a user can employ to build complex measurements:



Context-Aware Functions

Another distinction of the RI graphical programming environment is that blocks are context-aware. When function or measurement blocks' input/outputs are linked, the software can identify intrinsic characteristics of upstream and downstream blocks in the algorithm flow. This allows the test builder system to check continuity of variable type, data conversion, and unit of measurement of interconnected blocks internally, enabling the user to focus on measurement development.

Control Hierarchy



The programming process has a tiered system of global, section, and panel windows to help the user organize and maintain complex test plans and measurements. The "global" tier is used to define values and control settings for multiple TIMs and measurements that will the default if not declared in the lower tiers. The "section" tier enables the user to define multiple settings, variables, and algorithms that will apply to related tests or test instruments. The "panel" tier is used for implementing measurement algorithms, test commands and declaring variables for an individual test. For example, say Cassini was configured with 2 signal generators(Source1, Source2), and two receivers(Receiver1, Receiver2). The user's test plan called for a measure of IP3 and IMD. That translates to a sweep of a source power at a set frequency, a power measure at the receiver, and a frequency measurement using two source tones at a set frequency and power. Test equipment defaults shared by both tests would be placed in the Global: initial frequency and output power of each signal generator(may be different for each) and the tuned frequency for both receive paths. In the

“section” tier, each measurement has separate power and frequency limits/constraints that are specific to each test’s source and receive paths. In the “panel” tier, measurement blocks and defaults are configured to execute the individual IP3 or IMD tests, capture data, and display the results to the user. This resource organization structure gives the user the flexibility to exert control over several instruments simultaneously or individual instruments separately to suit the needs of their application.

Test Plan Simulator

The entire RI software platform was built using the smalltalk language and OS/2 operating system to minimize the delays and latencies related to a large operating system in favor of a more streamlined software platform. Consequently, the system’s small footprint lent itself to porting easily into a virtual environment on any user's native operating system. This allows RI to provide a simulator of the test system where a test plan developer can design, test, and debug their test plans on their local computer. With this debug development step in place, the user can verify their setup with an emulated TIM configuration before running it on the actual hardware to make use of limited time on the tester to take measurements and verify results.

Up Next

The next document in the series, "Optimization," discusses Cassini's unique automated test plan optimizer: RI Synapse. By using the architectural advantages of state-based programming, Synapse provides highly efficient test plan execution in hardware, dramatically reducing test time without adding to the user's design cycle.