

C# Reference Architecture



As device complexity continues to grow, developing test programs has become increasingly challenging. With shorter product life cycles and accelerated time-to-market demands, traditional manual coding approaches are no longer sustainable. New methodologies emphasizing reuse and automation are essential.

The **C# Reference Architecture** (C#RA) addresses these challenges by streamlining the creation and maintenance of test programs. It offers a robust, performance-oriented library of validated implementations that reflect recommended practice in test development.

The Library

At the heart of C#RA is a powerful abstraction layer known as [Test Blocks](#), which enables the development of concise, generic, reusable, and intuitive test code. These blocks are designed to be modular, high-performing, and immediately understandable - minimizing the need to consult documentation.

The provided [Test Methods](#) illustrate effective and proven solutions for common test scenarios, available in both offline and online runnable demo programs. By intentionally avoiding exhaustive coverage of every edge case, the examples remain clean and approachable - making them easy to adopt and extend.

This balance allows teams to address more specific requirements while maintaining a consistent and scalable development approach.

```
TheLib.Setup.LevelsAndTiming.Apply(true);
Services.Setup.Apply(setup);
if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
TheLib.Setup.Dc.Connect(_pins);

TheLib.Setup.Dc.SetForceAndMeter(_pins, TLibOutputMode.ForceCurrent, current, current,
clampVoltage, Measure.Voltage, voltageRange);
TheLib.Execute.Wait(waitTime);
_meas = TheLib.Acquire.Dc.Measure(_pins);

TheLib.Setup.Dc.Disconnect(_pins);
if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
TheLib.Datalog.TestParametric(_meas, current);
```

Comprehensive [documentation](#) complements the codebase, offering insights into the [requirements process](#), [design decisions](#) and providing [API help](#) for deeper understanding where necessary.

Infrastructure

C#RA is built with automation at its foundation. Automated unit and integration testing, documentation generation, and packaging workflows ensure consistent quality and reduce the risk of human error.

The project follows an agile development model, allowing for rapid iteration and continuous delivery through monthly releases.

These capabilities are extended to users, enabling seamless integration into their own workflows - supporting both quality assurance and timely delivery.

Getting Started

1. Download and unzip the latest [release](#)
2. Review the documentation: [/doc/index.html](#) or [online](#)
3. Understand the system requirements and prerequisites: [Documentation > Getting Started](#) or [online](#)
4. Explore the included demo program: [/Demo](#) or [online](#)
5. Integrate C#RA into your test project: [Documentation > Getting Started > Installation](#) or [online](#)
6. Start building better test programs - faster and more reliably!

Releases

Release Version	Publish Date
v0.13	2025-10-27
v0.12	2025-10-17
v0.11	2025-10-10
v0.10	2025-10-02
v0.9	2025-09-26
v0.8	2025-09-01
v0.7	2025-08-01
v0.6	2025-07-01
v0.5	2025-06-02

Release Version	Publish Date
v0.4	2025-05-09
v0.3	2025-04-01
v0.2	2025-03-10
v0.1	2025-02-21

Getting Started

Welcome to the official documentation for C#RA, a C# library designed to make the life of a test engineer easier by wrapping multiple information into simple code lines. This guide will help you get started quickly by walking you through downloading, installing, referencing, using, and debugging the library.

What I need to get started

- Access to eKnowledge
- A working IG-XL / .NET environment - see documentation or training for details.
 - 11.00.00 (Release)
 - 10.00.01 (Patch)
- Oasis Tool
 - 4.5.10501

Download & Extract

To begin using the C#RA library, you'll first need to download and extract the package. This package includes the core source code, a demo project to help you get started quickly, and offline documentation for reference. Follow the steps below to set up your environment.

- Visit the [C#RA page](#) on eKnowledge.
- Download the latest C#RA package.
- Unzip the contents to a location of your choice.

Content of the ZIP:

```
└── └── Csra # Source code of the library  
└── └── Demo # Sample project demonstrating usage  
└── └── docs # Offline HTML documentation
```

i TIP

You can create a Git repository for the `Csra` folder and link it as a submodule in your main project repository.

Demo Project

For reference purpose a demo project is part of the release package. It shows how to use a `TestMethod` or a `TestBlock` from a 3rd party project.

```
└── └── Demo  
    └── └── ASCIIProgram # IG-XL files  
    └── └── Demo_CS # Sample project demonstrating c# usage without C#RA  
    └── └── Demo_CS_DSP # Sample project demonstrating c# dsp usage without C#RA  
    └── └── Demo_CSRA # Sample project demonstrating c# usage with C#RA  
    └── └── Patterns # Pattern files  
    └── _LoadDemoProgram.cmd # Generate Visual Studio Solution and loads IG-XL via IGLinkCL  
    └── Demo.igxlProj # IG-XL project  
    └── SimulatedConfig.txt # For simulating purposes
```

To start IG-XL, just click on the `_LoadDemoProgram.cmd` file. Once IG-XL is started, open `Datalog` window and hit `Validate Job` and `Run` at the `IG-XL Main` tab.

To debug the Test Program, click `Visual Studio` (Connected Mode) at the `IG-XL Tools` tab. Once Visual Studio is up, add breakpoints and hit `Run` again.

Installation & Adding Reference

Once you've downloaded and extracted the Csra package, the next step is to integrate it into your development environment. Whether you're working in Visual Studio or using IG-XL, the process is straightforward. Below are the instructions for both environments to help you get up and running quickly.

- Copy the `Csra` folder from the downloaded package.
- Paste it next to your solution file.

Visual Studio Reference

To include the Csra project in your Visual Studio solution:

- Open your solution in Visual Studio.
- Right-click the solution > **Add > Existing Project...**
- Navigate to the Csra folder and select `Csra.csproj`.
- Click **OK**.

IG-XL Reference

To reference the compiled Csra library in IG-XL:

- Build the `Csra` project in Visual Studio to generate the `Csra.dll`.
- Open IG-XL.
- Navigate to `ReferenceSheet`.
- Right-click on the next empty cell, **Edit**.
- Navigate to the output folder and select `Csra.dll`.

NOTE

Adding a reference in IG-XL is only necessary if you plan to use **TestMethods** provided by the Csra project.

Use Test Method

Before using a Test Method from the Csra library, make sure you've completed the [IG-XL Reference](#) step. This ensures that IG-XL can access the compiled Csra.dll and recognize the available TestMethods.

- Open **IG-XL**.
- Navigate to the **Test Instance Sheet**.
- Enter the desired Test Method name (e.g., **Csra.Continuity.Parametric.Parallel**).
- Provide the required parameters for the selected method.
- Run **IG-XL** to execute the test.

 **NOTE**

The available TestMethods are defined in the Csra project. Make sure to consult the documentation or the source code to understand the expected parameters and behavior.

Required using Statements in Test Code

using Csra;

Purpose:

This directive is essential for accessing the core objects and functionality provided by the Csra framework.

Enabled Access To:

- Pins object
- PatternInfo object
- Setup object

When to Use:

- Required in nearly all test scenarios involving pin configurations

using Csra;

Purpose:

Provides access to version-specific features, including settings and test block definitions.

Enabled Access To:

- Setting object
- Extension methods
- Enums for TestBlocks
- Access to the Api object

When to Use:

- When you need to configure SetupService
- When working with TestBlocks

using static Csra.Api;

Purpose:

A convenience directive that allows direct access to static members like Services and TheLib without needing to qualify them.

Enables Access To:

- Services
- TheLib

When to Use:

When you want cleaner syntax for accessing static members.

NOTE

This is a shortcut and does not replace the need for `using Csra;`.

Summary & Best Practices

Directive	Required For	Notes
<code>using Csra;</code>	Core objects like Pins, PatternInfo, Setup, Settings, enums	Always needed
<code>using static Csra.Api;</code>	Convenience for static access	Optional, but useful

Why Multiple Usings?

Due to **versioning**, **interface/implementation separation**, and **modular design**, no single `using` directive can cover all use cases. This separation ensures flexibility and maintainability across different versions and components of the Csra framework.

Debugging C#RA

Before diving into debugging the C#RA project, it's highly recommended to review the [official guide](#) on debugging C# projects. This guide covers essential techniques and best practices that apply directly to C#RA and similar libraries.

 **NOTE**

The official guide provides comprehensive instructions, so this section does not repeat those details. Refer to it for step-by-step debugging help.

Suggest a Documentation Fix

Found a typo, outdated info, or something unclear in the documentation? You can easily suggest improvements directly through GitHub!

Each page on our documentation site includes an "Edit this page" button—usually located at the top or bottom of the page. Clicking it will take you to the corresponding Markdown file on GitHub, where you can:

- Propose edits directly in your browser.
- Submit a **Pull Request (PR)** with your suggested changes.

 **TIP**

You don't need to clone the repository or install anything locally - just a GitHub account is enough to contribute.

 **NOTE**

All contributions are reviewed before being merged, so feel free to suggest even small improvements. Every fix helps.

Request a Feature

Have an idea to improve the Csra library or its documentation? We'd love to hear it!

We use **GitHub Issues** to track all feature requests, bug reports, and enhancements. To suggest a new feature:

- Go to the GitHub [Issues page](#).
- Click **New Issue**.
- Describe your idea clearly.
- Submit the issue.

NOTE

Every submission is reviewed in the short term. If the feature is accepted, it will be assigned to a team member and tracked through GitHub.

TIP

You can also use issues to report bugs or request improvements to existing functionality.

Use Test Method

TBD

Create Custom TestBlock

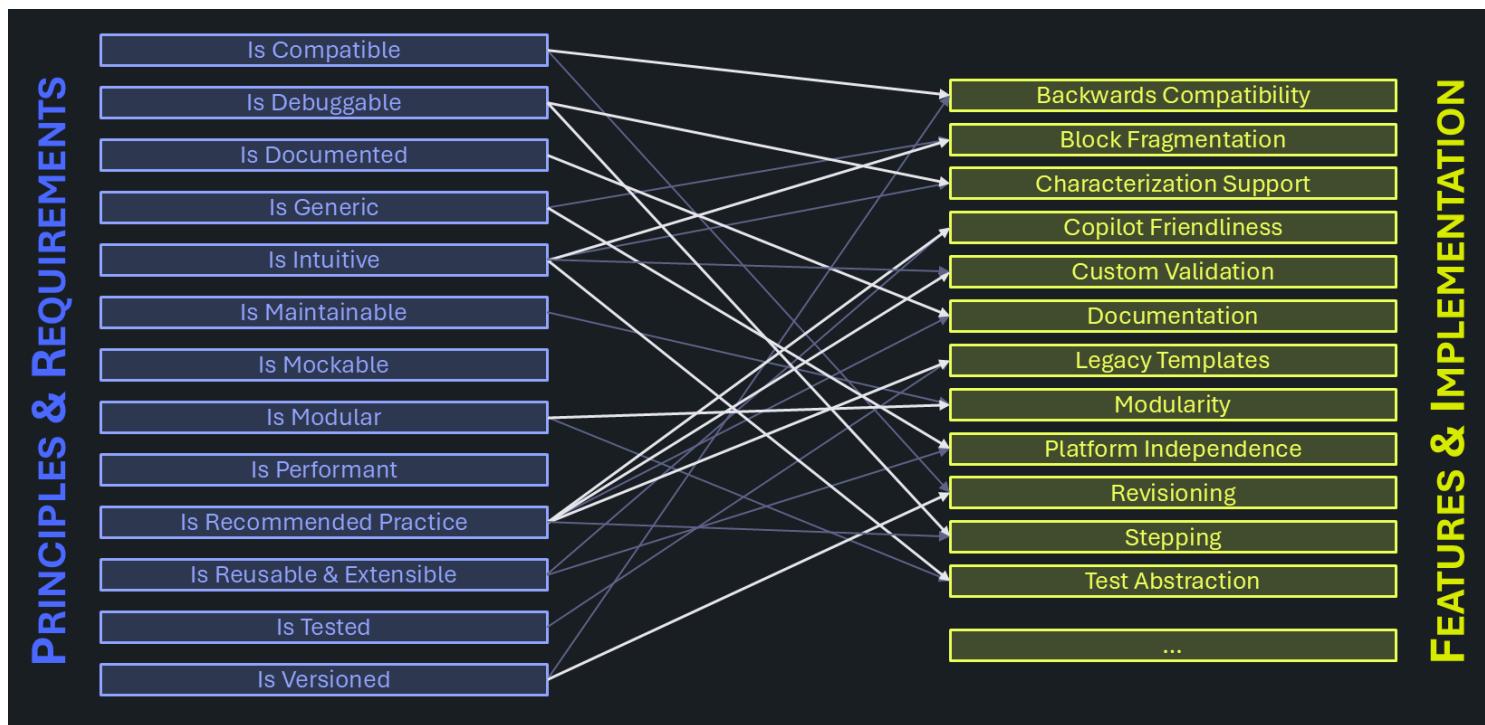
TBD

Features & Implementation

Derived from principles and requirements, a comprehensive list of features comprise the backbone of the **C# Reference Architecture**.

Design decisions made are documented with alternative options stated. The decision making process is captured, so that when requirements change, the logic chain can be revised to confirm choices made are still the best option on the table.

The feedback loop is closed by comparing the results (features) with the postulated principles and requirements derived from them.



Clean

- [Alert Service](#) - centralized messaging and exception handling
- [Behavior Service](#) - when logic gets a personality
- [Code Structure](#) - organize the source code entities
- [Custom Validation](#) - improve usability through argument checking
- [Documentation](#) - one-stop-shop for information seekers
- [Extensibility](#) - embrace incompleteness
- [External Libraries](#) - tap into others' IP
- [Instrument Specific Features](#) - access to specialized hardware benefits
- [MethodHandle Type](#) - delegate the work
- [Modularity](#) - users can mix and match
- [Multi-Target IG-XL Support](#) - flexibility for different versions of IG-XL
- [Offline Features](#) - upvalue engineering time

- [Persistent Data Storage](#) - save and reuse information within and beyond job runs
- [Pins type](#) - context aware objects for pin lists
- [Platform Independence](#) - remove friction and enhance fluidity between different testers
- [Search & Trim](#) - a straightforward approach to a challenging test technique
- [Setup Service](#) - manage device & tester setups and automate efficient transitions between
- [Single- & MultiCondition](#) - flexibility for uniform and specific test conditions
- [SSN](#) - Streaming Scan Network, Siemens' next gen Scan Technology
- [Test Abstraction](#) - reduce complexity by encapsulating details
- [Transaction Service](#) - being fluent in devices' dialects
- [Unit- & Integration Testing](#) - control quality and minimize defects
- [Versioning & Compatibility](#) - manage how functionality develops over time

Work in Progress

- [Characterization Support](#) - easily determine margins and process capability
- [Copilot Friendliness](#) - great output requires decent input
- [Legacy Templates](#) - a successful use model re-imagined
- [PatternInfo type](#) - context aware objects for pattern & vector data
- [Stepping](#) - Pre / Body / Post flow debug

Code Structure

Consistency in code structure and architecture can be beneficial for readers and help implementers avoid mistakes: a concept carefully designed, reviewed and agreed may serve as a template for similar features, so that fundamental questions don't need to be answered over and over again.

For the user, such a code base has a solid look and feel, and can be efficiently used after a quick learning curve. To achieve that, implementers need to hold back with personal flavors and give up some creative freedom.

The following basic entities exist in the C# Reference Architecture, with each having their own specifics, while still following a common style:

Entity	Description
Test Blocks	stateless methods combining common use cases for a higher abstraction language
Test Methods	callable from the Instance / Flow sheet in IG-XL
Test Classes	grouping containers for alike Test Methods with common data structures or sub functions
Services	singletons for global accessibility (typically stateful)
Types	dedicated containers for data & associated functionality to be used in test classes, test methods and test blocks
Enums	predefined sets of named values representing distinct states, categories, or options
Extension Methods	methods that add new functionality to existing types without modifying their original implementation.

Code Structure - Test Blocks

Test Blocks are the main feature of this project, providing an abstraction layer in how users interact with the IG-XL based (future also other!) testers. They introduce a whole new use model with a learning curve users have to master first in order to become productive customizing and crafting their own test methods.

That experience and any created work can not be jeopardized by frequent, incompatible changes. Compatibility is a concern and is addressed so that existing code doesn't break, but updates and improvements are possible once the user understands and embraces rework requirements.

The goal for the Test Block API is a structured and user-friendly tree model similar to nested static classes. The main objectives are to ensure that the API is mockable, extendable, and easy to use.

A key requirement is the ability to support multiple API versions. This feature maintains backward compatibility while allowing users to adopt new functionality without affecting existing implementations.

Test Blocks fully implement [versioned interfaces](#) and are exclusively called through these. Previous versions are part of the release package and can keep being called. Implementation is kept common as far as possible.

(**IMPORTANT**

Because of their purely functional and stateless nature, test blocks may be called from other test blocks. This can be to avoid code duplication as well if there are superset designs, and may be required if private support methods are not sufficient. Also in the case of calling test block from other test blocks, calls are being made through their (versioned) interface. Once a new (incompatible) version is introduced, the C#RA team commits to do due diligence and update all calls to the then-newest version.

Language Hierarchy

The test block calling syntax is designed to be both intuitive and self-documenting. The language object conveys user intent — from broad, high-level domains to detailed, specific instructions — which makes the flow and sequence of code within a test method easy to follow. This clarity not only helps readers understand other's code but also assists developers in navigating a large and potentially unfamiliar API. The level depth adapts to the complexity of the branch or the specificity of the block method:

1. **Entry point:** `TheLib`
2. **Action** category: `Setup`, `Acquire`, `Execute`, `Datalog` ...
3. **Domain** category: `Dc`, `Digital`, `Ac`, `Rf` ... or test block methods

4. Further category - or test block methods

5. ...

Code Architecture

Two architectural approaches are considered: a static tree structure and a tree of multiple singletons. In both cases, the API tree is separated from the method implementations. The API consists of branches that reference a separate implementation section, keeping the API structure distinct from its functional logic. This separation improves modularity and maintainability.

The findings, the reasoning behind the chosen approach, and the key design trade-offs are listed here.

Static Approach

The static approach is pretty straight forward. It is possible to place all of the API in a single class file or separate into multiple files with partial classes.

```
namespace Csra.V1 {  
    public static partial class TheLib {  
        public static partial class Acquire {  
            public static void Meter() => Implementation.AcquireBlocks.Meter();  
        }  
        public static partial class Setup {  
            public static partial class Dc {  
                public static void Connect(string pins, bool gate) =>  
                    Implementation.Setup.DcBlocks.Connect(pins, gate);  
                public static void ForceI(string pins, double current) =>  
                    Implementation.Setup.DcBlocks.ForceI(pins, current);  
                public static void ForceV(string pins, double voltage) =>  
                    Implementation.Setup.DcBlocks.ForceV(pins, voltage);  
            }  
            public static void ApplyLevelsTiming() =>  
                Implementation.SetupBlocks.ApplyLevelsTiming();  
        }  
    }  
}
```

Singleton Approach

The singleton approach is a bit more complicated. The singleton approach uses nested interfaces that are declared in a separate file. In the next chapter the two approaches will be compared.

```
namespace Csra.V1 {
```

```

public static class Api {
    private static ILib _theLib = null;

    public static void MockInjection(ILib mockedObject) => _theLib = mockedObject;

    public static ILib TheLib => _theLib ??= new TheLib_();
    public static ILib.ISetup Setup => TheLib.Setup;
    public static ILib.IAcquire Acquire => TheLib.Acquire;

    private class TheLib_ : ILib {

        private static Setup_ _setup = null;
        private static Acquire_ _acquire = null;

        public ILib.ISetup Setup => _setup ??= new Setup_();

        public ILib.IAcquire Acquire => _acquire ??= new Acquire_();

        private sealed class Setup_ : ILib.ISetup {
            private static Dc_ _dc = null;
            public ILib.ISetup.IDc Dc => _dc ??= new Dc_();
            public void ApplyLevelsTiming() =>
                Implementations.TestBlocks.SetupBlocks.ApplyLevelsTiming();
            private sealed class Dc_ : ILib.ISetup.IDc {
                public void Connect(string pins, bool gate) =>
                    Implementations.TestBlocks.Setup.DcBlocks.Connect(pins, gate);
                public void ForceI(string pins, double current) =>
                    Implementations.TestBlocks.Setup.DcBlocks.ForceI(pins, current);
                public void ForceV(string pins, double voltage) =>
                    Implementations.TestBlocks.Setup.DcBlocks.ForceV(pins, voltage);
            }
        }
        private sealed class Acquire_ : ILib.IAcquire {
            public void Meter() => Implementations.TestBlocks.AcquireBlocks.Meter();
        }
    }
}

}

namespace Csra.Interfaces {

public interface ILib {
    public ISetup Setup { get; }
    public IAcquire Acquire { get; }
    public interface ISetup {
}

```

```

public IDc Dc { get; }
public void ApplyLevelsTiming();
public interface IDc {
    public void Connect(string pins, bool gate);
    public void ForceV(string pins, double voltage);
    public void ForceI(string pins, double current);
}
}
public interface IAcquire {
    public void Meter();
}
}
}

```

Comparison Goals

To effectively compare the two architectural approaches, I established several key goals that the API must meet:

- **Easy to Use** - The API should support efficient traversal of the object tree, allowing users to navigate through the hierarchical structure with ease.
- **Namespace Versioning Concept** - Different version of the API can coexist and be selected as needed. Without having too much overhead.
- **Extensible** - The API should be designed to allow for easy extension, like Extension-Methods.
- **Mockable** - The API should be easily mockable to facilitate unit testing and ensure that different components can be tested in isolation.

Easy to Use

Both approaches allow a nice way of calling methods in the API tree. The debugging experience is exactly the same for both approaches. The entry node of the singleton approach is the only difference.

```

# static
TheLib.Setup.Dc.ForceV("dig", 2.9);

# singleton
Api.TheLib.Setup.Dc.ForceV("dig", 2.9);

```

To overcome this syntax issue, it is possible to use static import, it allows you to access static members of a class without needing to qualify them with the class name.

```
# singleton with static import
using static Csra.Api;
...
TheLib.Setup.Dc.ForceV("dig", 2.9);
```

Namespace Versioning Concept

Both approaches are again very similar to achieve the goal of namespace versioning. It is quite easy to select between versions, by file or even by line of code.

```
# static
using Csra; // selects the default version for the file
...
TheLib.Setup.Dc.ForceV("dig", 2.9); // executes ForceV of V1
...
Csra.V2.TheLib.Setup.Dc.ForceV("dig", 2.9); // executes ForceV of V2
```

```
# singleton
using static Csra.Api; // selects the default version for the file
...
TheLib.Setup.Dc.ForceV("dig", 2.9); // executes ForceV of V1
...
Csra.V2.Api.TheLib.Setup.Dc.ForceV("dig", 2.9); // executes ForceV of V2
```

Extensible

If a customer wants to extend the capability of the C# Reference Architecture, it should be far easy to extend methods. The static approach allows you to write custom methods in a separate file **but** inside the cs-reference-architecture project. There could be issues when trying to update cs-reference-architecture in the future.

```
# static
## Needs to be in the same project as the API, but not in the same file
namespace Csra.V1 {
    public static partial class TestBlock {
        public static partial class Setup {
            public static partial class Dc {
                public static void CustomMethod(string pins) {
                    // This is doing custom things
                }
            }
        }
    }
}
```

```

    }
}

## it can be used like
TestBlock.Setup.Dc.CustomMethod("hi");

```

Using ExtensionMethods adds the ability to define those methods in a separate project but make it accessible through the API.

```

# singleton
## can be in the customer project
public static class Extensions {
    public static void CustomMethod(this Csra.Interfaces.ILib.ISetup.IDc dc, string pins) {
        // This is doing custom things
    }
}

## it can be used like
using static Csra.Api;
...
TheLib.Setup.Dc.CustomMethod("hi");

```

Mockable

The cs-reference-architecture is unit-tested, a customer will use those Test Blocks in their custom code. User written unit-test code should not depend on the C# Reference Architecture and therefore mocking those calls is required.

While the static approach does not support to be mocked, the singleton approach supports those almost out of the box. The MSTest project needs to import [Moq](#) to mock C# Reference Architecture.

```

#singleton
#CustomerCode
public void DoWhatYouWant(string pins, bool gate) {
    if (gate) {
        // do custom things
    }
    TheLib.Setup.Dc.Connect(pins, gate);
    if (TheHdw.DCVS.Pins(pins).Gate) {
        // do other things
    }
}
...
#endif

```

```

[TestMethod]
public void TestDcConnectMock() {
    Mock<Csra.Interfaces.ILib> mockTestBlock = new Mock<Csra.Interfaces.ILib>() {
DefaultValue = DefaultValue.Mock };
    MockInjection(mockTestBlock.Object); // Inject the mocked object, so TestBlock is using
the mocked object instead of the real deal
    DoWhatYouWant("dig", false);
    mockTestBlock.Verify(x => x.Setup.Dc.Connect("dig", false), Times.Once); // make sure
that Setup.Dc.Connect was called with these arguments
    // Assert custom things
    // Assert other things
    MockInjection(null); // cleanup mocking after the TestMethod
}

```

This gives the user the ability to test their custom code isolated from the C# Reference Architecture.

Conclusion on Architecture

The singleton approach has the capability of mocking which the static does not have and supports way nicer extension methods. The versioning and object tree traversal are fairly similar. Consequently, the singleton approach is selected.

Instrument Independence & Feature Tolerance

Instead of programming instrument features directly, test blocks use a `TheLib.Action.Domain` notation, with the domain relating to generic instrument capabilities rather than types. For example most of the instrument have some basic DC capability, and that way this can be controlled commonly with a single call.

Inside the block, driver calls have to be routed to the specific instrument language nodes in IG-XL, and the `Pins` type helps extracting the relevant pins for these in an efficient way.

Principles for Test Blocks

1. Test Blocks tolerate pins that don't offer the feature required. They will extract the sub-pin list required to program the hardware and quietly ignore others. This allows for simpler and instrument / platform agnostic code at the test method level. Not requiring runtime checks for "any pins left over" improves execution performance.
2. Test Method authors are advised to make use of validation to check that the provided pins support the expected type and no pins of unsupported types are provided. C#RA provided Test Methods lead by example and implement that.
3. Because correct parameter validation can't be guaranteed in user code, test blocks issue an `Services.Alert.Warning` if the provided pin list doesn't contain any supported pins. This would

result in quietly performing no action at all, which is considered an untypical and potentially dangerous case which should be highlighted. This check can be implemented in test blocks at little execution time cost.

4. Scenarios that do require this case may suppress the warning with a conditional call of the test block. The `if (_pins.ContainsFeature(Pins.Feature.Digital))` `TheLib.Setup.Digital.Disconnect(_pins)` both prevents that warning and documents the intention in the code. The concept is similar to the `pragma` compiler statements, only that these don't apply to run-time.

Implementation

A block only supporting a single instrument type would extract the relevant pins to perform a driver call and issue the warning in the `else` path:

```
internal static void Connect(Pins pins) {  
    if (pins.ContainsFeature(Pins.Feature.Digital, out string pinList)) {  
        TheHdw.Digital.Pins(pinList).Connect();  
    } else {  
        Services.Alert.Warning("None of the pins contain 'Digital' features - no action performed", "Setup");  
    }  
}
```

Blocks supporting different instrument types need a little extra logic to determine that case:

```
internal static void ForceV(Pins pins, double forceVoltage) {  
    bool noAction = true;  
    if (pins.ContainsFeature(Pins.Feature.Ppmu, out string ppmuPins)) {  
        TheHdw.PPMU.Pins(ppmuPins).ForceV(forceVoltage);  
        noAction = false;  
    }  
    if (pins.ContainsFeature(Pins.Feature.Dcvi, out string dcviPins)) {  
        TheHdw.DCVI.Pins(dcviPins).Voltage.Value = forceVoltage;  
        noAction = false;  
    }  
    if (pins.ContainsFeature(Pins.Feature.Dcvs, out string dcvsPins)) {  
        TheHdw.DCVS.Pins(dcvsPins).Voltage.Value = forceVoltage;  
        noAction = false;  
    }  
    if (noAction) {  
        Services.Alert.Warning("None of the pins contain 'DC' features - no action performed", "Setup");  
    }  
}
```

```
    }
}
```

The code for Test Methods remains simple but adds explicit highlighting of (legitimate) scenarios where no-action test blocks are supported:

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class Parallel : TestCodeBase {

    private Pins _pins;
    private PinSite<double> _meas;
    private PatternInfo _pattern;
    private bool _digitalPresent;

    /// <summary>
    /// Measures the current at the bias voltage applied to the pins of a device.
    /// </summary>
    /// <param name="pinList">List of pin or pin group names to measure.</param>
    /// <param name="voltage">The force voltage value.</param>
    /// <param name="currentRange">The current range for measurement.</param>
    /// <param name="waitTime">The wait time after forcing.</param>
    /// <param name="setup">The name of the setup set to be applied through the setup
    service.</param>
    #region Baseline
    [TestMethod, Steppable, CustomValidation]
    public void Baseline(PinList pinList, double voltage, double currentRange, double
    waitTime, string setup = "") {

        if (TheExec.Flow.IsValidating) {
            _pins ??= new Pins(pinList);
            _digitalPresent = _pins.ContainsFeature(Pins.Feature.Digital);
            // add validation to check if any of the pins support DC
        }

        if (ShouldRunPreBody) {
            TheLib.Setup.ApplyLevelsTiming();
            Services.Setup.Apply(setup);
            if (_digitalPresent) TheLib.Setup.Digital.Disconnect(_pins);
            TheLib.Setup.Dc.Connect(_pins, true);
        }

        if (ShouldRunBody) {
            TheLib.Setup.Dc.ForceV(_pins, voltage, voltage, currentRange);
            TheLib.Execute.Wait(waitTime);
        }
    }
}
```

```
_meas = TheLib.Acquire.Dc.Measure(_pins);  
}  
  
if (ShouldRunPostBody) {  
    TheLib.Setup.Dc.Disconnect(_pins, false);  
    if (_digitalPresent) TheLib.Setup.Digital.Connect(_pins);  
    TheLib.Datalog.TestParametric(_meas, voltage);  
}  
}  
}
```

 **NOTE**

The flag `_digitalPresent` may be determined at validation time, but isn't expensive even if called at run time. The pins type uses cached information on pin types and features, and may even internally cache this information going forward if profiling results indicate a benefit.

Validation checks are making sure the test method isn't called with pins it does not support (exact syntax to be determined).

Test Methods

Test Methods in C#RA are built from test blocks as a reference implementation for common use cases. They show recommended practice, serve as self-documenting examples and can directly be (re-)used if they fit in the target scenario.

They are however not trying to be everybody's darling with a superset of functionality to make them suitable also for corner cases or uncommon combination of requirements (that place is already taken by Template.xla). Instead, users are encouraged to "roll their own", taking peeks from our implementations, copy & paste code and customize to their needs. Preferably in building their own "library" of customer specific, but reusable test methods that work great for the device families and derivatives needed.

That approach helps preventing the test methods offered as part of the C# Reference Architecture from getting bloated and hard to maintain. They are updated as IG-XL progresses and features evolve, so that they are a role model how to best use the product. Incompatible updates are avoided, but if inevitable, the changes are well and pro-actively documented.

Being called from the test instance sheet, a few special conditions apply to IG-XL test methods:

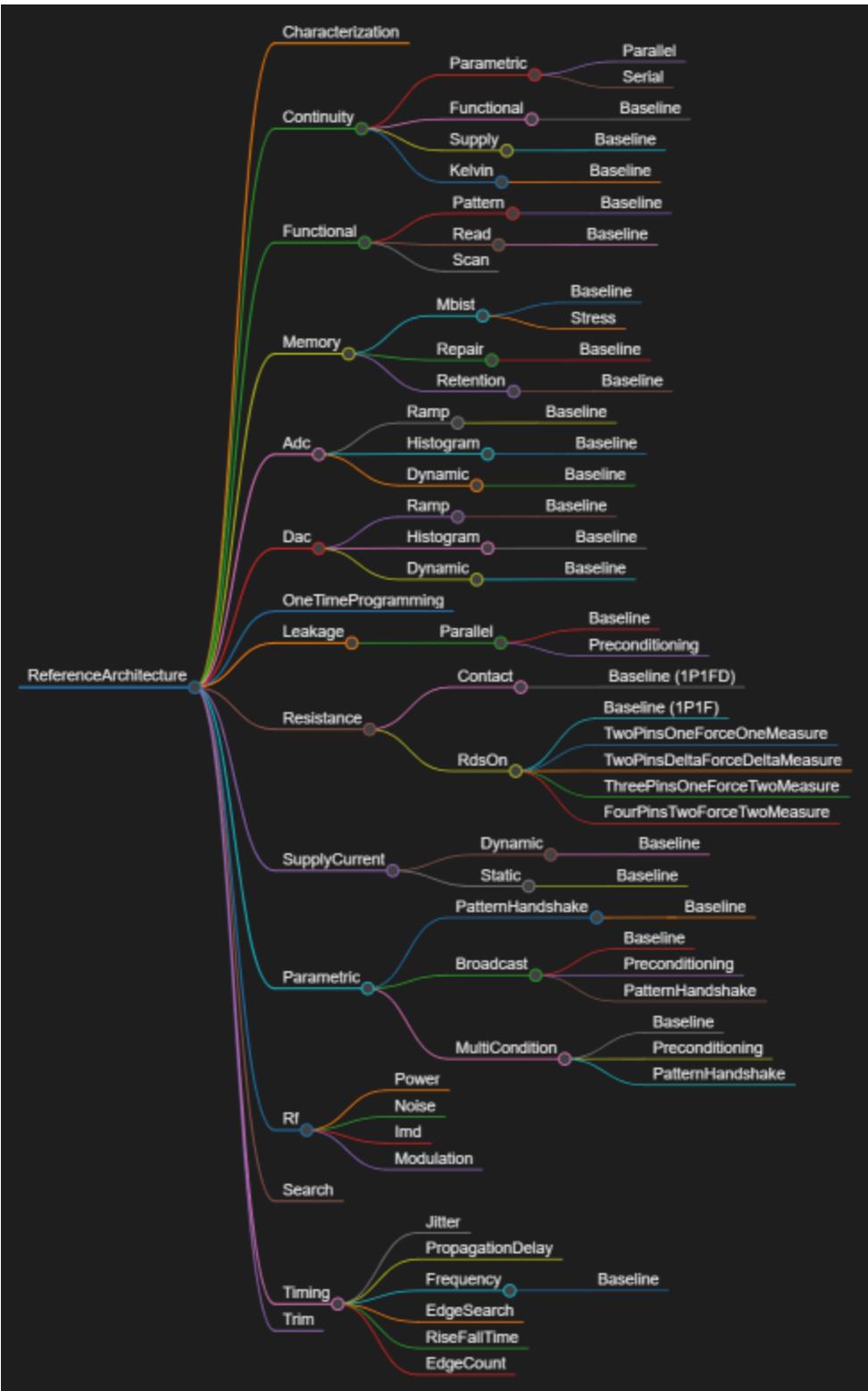
- need to be `[TestMethod]`s inside `[TestClass]`es
- can't use overloads
- fully qualified name always visible on the instance sheet (no concept of using exists)
- no type-ahead, only a drop-down to browse in alphabetical order
- not called from within code (exclusively from flow / instance sheet)

Test Instances		
	Test Procedure	
Test Name	Type	Name
Cont	.NET	ReferenceArchitecture.Continuity.Parametric.Parallel
Cres	.NET	ReferenceArchitecture.Resistance.Contact.Baseline
Iddq	.NET	ReferenceArchitecture.SupplyCurrent.Dynamic.Baseline
GndOffset	.NET	ReferenceArchitecture.Parametric.Broadcast.Preconditioning

Test Methods are not using versioned interfaces - only the latest versions are part of the release package. Previous (incompatible) versions may accessed from user code if needed, following the regular process of customizing our reference implementations.

Language Hierarchy

In order to minimize friction for users locating test methods they need, and for authors to offer a clear place to add new ones, the following hierarchy is used as [defined & maintained here](#):



Test methods are **always** placed in a four-level hierarchical structure along this scheme:

1. Entry Point
2. Test Category
3. Test Class
4. Test Method

Entry Point

The entry point is called **Csra**.

All test methods offered by the C#RA are found under this node.

Test Category

The first level grouping reflects device block or test strategy related groups commonly found in test programs. Often, development is broken down to these functional concepts, dedicated team members with specific experience implement and use sub-flows following this structure.

Prominently providing this grouping can help readers or other non-authors involved with the test program as they can see the intention ("leakage" vs. "functional-parametric"), even though such grouping may result in overlap and redundancy in the offerings. The C#RA team opted for the not-so-easy path to improve usability and readability.

It is understood that this categorization will never be 100% clear and perfect, with categories being cross-cutting and blurring test techniques, methodologies or features. It was however the best of all other alternatives considered.

For new branches, subject matter experts are consulted to create a meaningful scheme fitting into the other categories (for their peers **and** for those who won't use it). Overall, it is considered okay if the number of test categories grows, but excessiveness is to be avoided. Similarly, the various domains should be consistently represented, with neither flooding the space.

Test Categories are implemented as sub namespaces.

Test Class

Test classes resemble the next level of grouping. They are named to describe the commonalities of the contained test methods but distinctive enough to set them apart from the others in the same test category. The name of that isn't repeated in the class name, even if that ends up being rather generic or ambiguous (**Read**) - it will only be visible on the instance sheet as part of the fully qualified name together with the test category.

Because IG-XL creates test class objects for the execution of a test method, the class footprint matters. They avoid large numbers of test methods or little functional or algorithmic overlap in the implementation. Class level fields should typically sharable between the test methods, or they are supersets.

When in doubt, prefer separate, more specific test classes. The number of test classes within a test group is uncritical, but consider alphabetical sorting when offering similar ones that only differ in a detail (**SupplyAbc** & **SupplyDef** is preferred over **AbcSupply** & **DefSupply**).

Even though technically possible in C#, test classes don't repeat the name of their parent test category (namespaces).

Test Method

Test methods finally are the entities called from the flow. Some test concepts / classes will have few or even only a single test method, while others require multitudes. The name reflects the specific purpose but is brief and clear. Multiple test methods clearly differentiate their functionality by their names.

Baseline is used for test methods that cover many (most) use cases and have no more descriptive, industry known name. In the case of more than one test methods in a test class, **Baseline** refers to the clearly dominant & common, or the most straight-forward use case. Others are named so that they clearly indicate how they differ from that ([Leakage.Parallel.Baseline](#) & [.Preconditioning](#)).

Alternatives Considered

Originally, a flat hierarchy was considered without adding a sub-namespace. It became quickly clear that the necessary grouping to avoid ambiguities would then sneak into the test class names, saving no screen space but giving away an opportunity to create structure in the implementation.

The following alternatives for method names **Baseline** were rejected:

- **Base** - too close to inheriting from base classes
- **Basic** - could be confused with VisualBasic
- **Default** - means using something that is explicitly specified elsewhere
- **Simple** - incorrect as the common case might not be the simple one

Open Items

The topic of versioning in test methods was not finally decided. The following thoughts are on the table:

- multiplying test methods for all versions will fill up the instance drop-down further
- name gets even longer if version is thrown in (although only few characters)
- users want the latest? "I install a new version for a purpose - of course I want all test methods to benefit"
- users want compatibility? "Upgrading to a new library allows you selectively access new content, but you maintain compatibility for everything that already exists"
- Offering all previous versions is our answer to the usability degrade dilemma as users expect compatibility. How would we solve that without offering versioned test methods?
- can we use a formula in test method names?
- any other idea to improve the use model? Including potential IG-XL features?

Code Architecture

| to be defined

Code Structure - Services

Services are centralized, stateful feature providers that can be globally reached from user code, test methods, blocks, types or other services. They are an integral part of the overall use model reflecting recommended practice to use the tester.

As such, their use model and functionality is being relied on in user code, with significant impacts in the case of incompatible changes. Similar to Test Blocks, C#RA offers a way to keep existing code running, while allowing use model improvements to be offered.

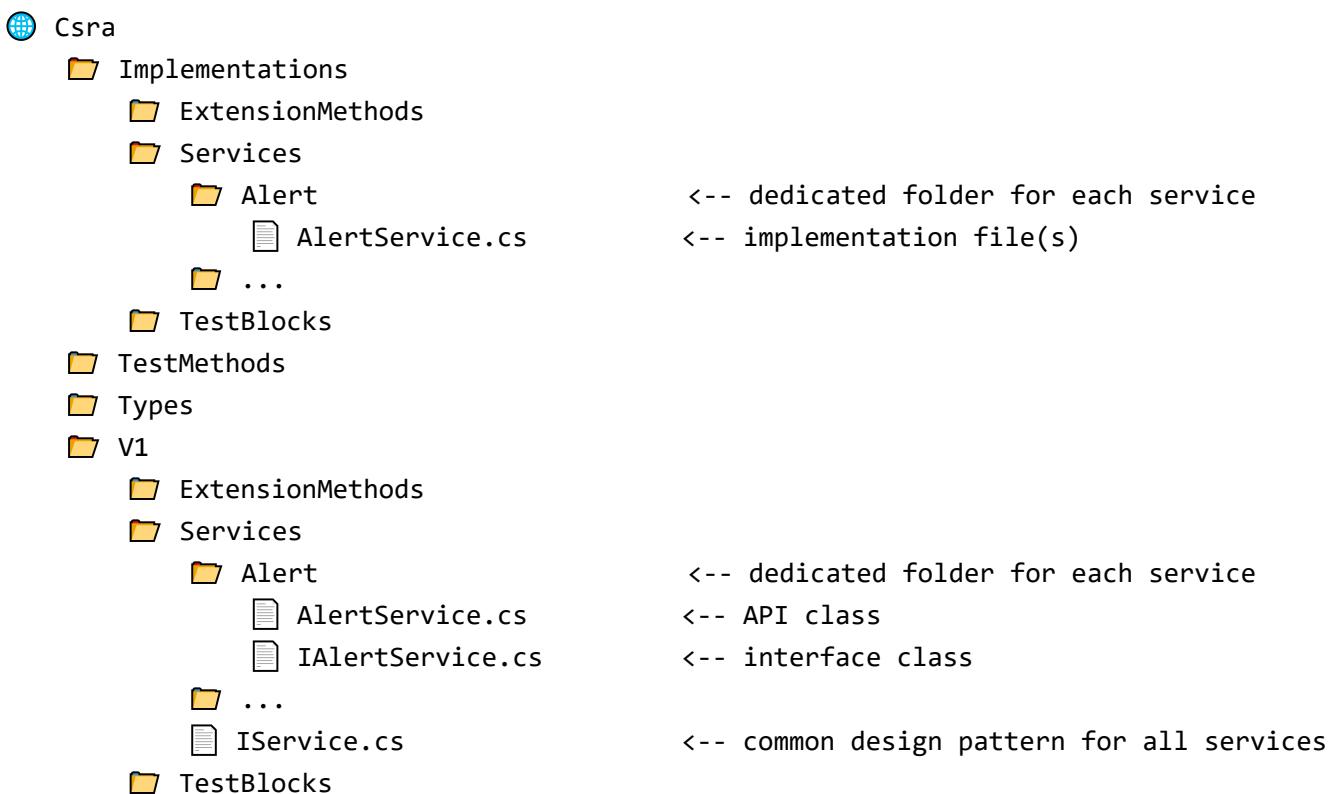
Services are exclusively accessed via versioned interfaces. Multiple versions are available in a release package, allowing existing code to keep running if pointing to a version before an incompatible update. They share common implementation, realized as a static class.

IMPORTANT

Calls to Services from within the C# Reference Architecture always use the latest interfaces, and are updated when a new (incompatible) version is introduced.

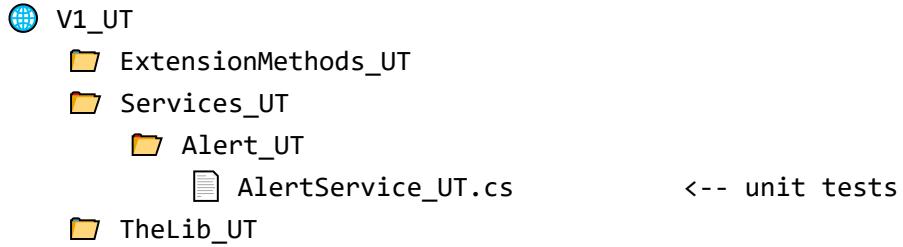
File & Folder Structure

All Services follow a consistent scheme. The following examples will use the [AlertService](#) as reference:



```
graph TD; Csra[Csra] --> Implementations[Implementations]; Implementations --> ExtensionMethods[ExtensionMethods]; Implementations --> Services[Services]; Services --> Alert[Alert]; Alert --> AlertServiceCS[AlertService.cs]; Alert --> IAlertServiceCS[IAlertService.cs]; Alert --> Ellipsis1[...]; Services --> TestBlocks[TestBlocks]; TestBlocks --> TestMethods[TestMethods]; TestBlocks --> Types[Types]; TestBlocks --> V1[V1]; V1 --> ExtensionMethods[ExtensionMethods]; V1 --> Services[Services]; Services --> Alert[Alert]; Alert --> AlertServiceCS[AlertService.cs]; Alert --> IAlertServiceCS[IAlertService.cs]; Alert --> Ellipsis2[...]; Services --> IServiceCS[IService.cs]; IServiceCS --> Ellipsis3[...]; Services --> TestBlocks[TestBlocks]
```

The diagram illustrates the folder structure for a service named "Alert". It shows a main "Csra" directory containing "Implementations", "TestBlocks", "TestMethods", "Types", and "V1". The "V1" directory contains "ExtensionMethods" and "Services". The "Services" directory for "V1" contains a "Alert" folder, which includes "AlertService.cs" and "IAlertService.cs". There are also ellipsis entries for "Services" and "TestBlocks". The "Implementations" directory contains "ExtensionMethods" and "Services". The "Services" directory under "Implementations" contains a "Alert" folder, which includes "AlertService.cs" and "IAlertService.cs". There are also ellipsis entries for "Services" and "TestBlocks". The "TestBlocks" directory contains "TestMethods", "Types", and "V1". The "V1" directory under "TestBlocks" contains "ExtensionMethods" and "Services". The "Services" directory under "V1" contains a "Alert" folder, which includes "AlertService.cs" and "IAlertService.cs". There are also ellipsis entries for "Services" and "TestBlocks". The "Implementations" directory under "Csra" contains "ExtensionMethods" and "Services". The "Services" directory under "Implementations" contains a "Alert" folder, which includes "AlertService.cs" and "IAlertService.cs". There are also ellipsis entries for "Services" and "TestBlocks".



Interface Class

The versioned interface definition holds all public nodes along with the XML documentation:

API Class

The versioned API is a partial class holding the singleton object and implements the interface. Calls are simply handed through to the implementation. Incompatible updates made there (like an extra parameter that was added for additional functionality in the implementation) can be disguised at this level, so that legacy versions remain functionally unchanged:

Implementation Class

The implementation class holds the actual functionality. It's a functional superset for all existing versions:

Common Design Pattern

To facilitate certain commonality across all C#RA services, they implement the `IService` interface:

```

namespace Csra.V1 {

    public interface IService {

        /// <summary>
        /// Initialize the service. This is called by the API when the service is
        first used.
        /// </summary>
        void Reset();
    }
}

```

At this point, only a common `Reset()` method is required. Besides the consistent use model, this helps avoiding state leakage in consecutive unit tests: all services are reset in test initialization.

Unit Testing

Since they provide central capability for a broad and diverse range of use cases, Services are extensively unit tested. Not only should line coverage be at 100%, but the goal is to really try all corner cases and combinations, so that users can rely on flawless functionality:

```
using Moq;
using System;
using System.IO;
using Csra;
using Teradyne.Igxl.Interfaces.Public;
using static Csra.Api;
using MsTest = Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Services_UT {

    [MsTest.TestClass]
    public class AlertService_UT : UT.Base {

        [MsTest.DataTestMethod]
        [MsTest.DataRow(0)]
        [MsTest.DataRow(AlertOutputTarget.OutputWindow)]
        [MsTest.DataRow(AlertOutputTarget.OutputWindow | AlertOutputTarget.Datalog)]
        [MsTest.DataRow(AlertOutputTarget.OutputWindow | AlertOutputTarget.File)]
        [MsTest.DataRow(AlertOutputTarget.OutputWindow | AlertOutputTarget.Datalog
| AlertOutputTarget.File)]
        public void LogTarget(AlertOutputTarget input) {
            AlertOutputTarget outputDefault = AlertOutputTarget.OutputWindow;

            Services.Alert.LogTarget = input;

            MsTest.Assert.AreEqual(input | outputDefault, Services.Alert.LogTarget);
        }
    }
}
```

Code Structure - Typed

Types are used to carry complex data between test blocks and to persistently store it in test methods. In an object oriented model, class objects (dedicated types) are pervasively used as arguments and return types of test blocks.

Public C#RA types are carefully designed for generic use and backwards compatibility. Required extensions and enhancements are made in ways not affecting existing implementations, incompatible updates are eschewed. Because of that liability, the introduction of any new custom types in C# Reference Architecture is carefully considered.

Adding and consistently supporting fully versioned version of types throughout the C# Reference Architecture would add significant complexity and code duplication to the project, to an extent where usability would be significantly thwarted.

Code Structure - Enums

Enums are value types assign speaking names to distinct states, categories, or options. Both, the individual members, as well as whether an option is available in a collection or not conveys important information.

As such, they help offering an intuitive use model, and any changes in versions need to be reflected there.

Public enums are offered and exclusively used in a versioned flavors where exposed to users. They map to internal superset enums used by the implementation, which addresses any compatibility tasks.

Implementation

On the API side, the enum is defined on the - versioned - interface. XML documentation provides context information for the user.

```
namespace Csra.V1 {  
  
    /// <summary>  
    /// The available output targets for Alert Service messages.  
    /// </summary>  
    [Flags]  
    public enum AlertOutputTarget { OutputWindow = 1, Datalog = 2, File = 4 }  

```

To avoid a direct dependency, the enum is defined again on the implementation side:

```
namespace Csra.Implementations.Services.Alert {  
  
    [Flags]  
    public enum AlertOutputTarget { OutputWindow = 1, Datalog = 2, File = 4 }  

```

Here, it needs to be a superset of features to match implementation also covering common functionality for all supported versions. Over time, these enums may drift apart, and careful design needs to make sure to avoid ambiguities or mismatches. Bi-directional explicit conversion in the API provide robust and type-safe translation:

```
public AlertOutputTarget LogTarget {  
    get => (AlertOutputTarget)Implementations.Services.Alert.AlertService.LogTarget;  
    set => Implementations.Services.Alert.AlertService.LogTarget =
```

```
(Implementations.Services.Alert.AlertOutputTarget)value;  
}
```

The cast here is fast, as no data conversion is involved. There is also no data check, so implementation has to provide safe default cases for seemingly non-existing enums - they might appear in the future and then break the code.

Overall however, this approach provided dedicated enum values per version without constraining incompatible changes (additions, removals and name changes) in case they become required.

Private enums, used only inside implementation without exposure to the user can be normally defined and used.

Code Structure - Extension Methods

Extension Methods are methods that add new functionality to existing types without modifying their original implementation.

Extension Methods are offered and exclusively used in a versioned flavor where exposed to users. They map to internal superset extensions used by the implementation, which address any compatibility tasks.

Implementation

On the API side, the extension is defined on the - versioned - interface. XML documentation provides context information for the user.

```
namespace Csra.V1 {  
  
    /// <summary>  
    /// Returns the single element of a sequence, or the element at the specified index if  
    /// the sequence contains more than one element.  
    /// </summary>  
    [DebuggerStepThrough]  
    public static T SingleOrAt<T>(this T[] values, int index) =>  
        Implementations.Extensions.SingleOrAt(values, index);
```

To avoid a direct dependency, the extension is defined again on the implementation side:

```
namespace Csra.Implementations {  
  
    internal static T SingleOrAt<T>(this T[] values, int index) => values.Length == 1 ?  
        values[0] : values[index];
```

Mocking extension methods directly is not possible because they are static methods. Most mocking frameworks, including Moq, do not support mocking static methods, which limits the ability to mock extension methods directly.

Alert Service

For a consistent user experience, centralizing messages in a common service is helpful: a common look and feel can be warranted, and changes need to be implemented only in a single place to update the entire system.

IMPORTANT

Alerts should be used with care to avoid overwhelming the user, and only be presented in extraordinary, special cases. Don't use the **Alert Service** for verbose mode logging of regular operation steps without offering a way to disable them.

A test program running in production mode where everything goes according to plan should not produce any alerts!

Info Alerts

Informative alerts are intended for messages with a positive / neutral information content. Anything relevant that should be highlighted to the user, for example a special operation mode, even if it was deliberately selected.

```
Services.Alert.Info("Test time profiling active");  
  
// INFO: Test time profiling active [OnProgramStarted]
```

Info alerts are labeled with **INFO:** and have the calling method listed at the end. Black color / regular style font is used in the OutputWindow.

NOTE

Basic **CallerMemberName** tracing allows the retrieval of a calling method's name at zero overhead cost. More thorough stack analysis would allow showing the fully qualified name of the method and other, potentially useful information, but at a cost of ~10ms per call this was considered too expensive for broad use.

Warning Alerts

Use warnings for non-fatal, but likely problematic or unexpected situations that can be recovered. An example would be the lack of calibration factors, so that defaults have to be assumed.

```
Services.Alert.Warning($"calibration file '{calFile}' not found - using defaults");  
  
// WARNING: calibration file 'cal.json' not found - using defaults [ImportCalData]
```

Warning alerts are labeled with **WARNING:** and have the calling method listed at the end. Dark blue color / regular style font is used in the OutputWindow.

Warnings do not trigger any other further action, like propagating the state to IG-XL, turn off any instruments, end flows or bin a device. From an IG-XL perspective, warnings are no different than info alerts.

Error Alerts

Error alerts should be used for fatal, non-recoverable situations that disallow further, possibly dangerous execution of a test program.

In normal program runs, Error alerts are logged and then propagated to IG-XL by throwing an exception and having that propagate to the IG-XL Error service, following the recommended practice for .NET test code. IG-XL will handle the error, and bin out devices correctly as set up. When issued during validation, the IG-XL validation service is notified, resulting in validation to fail.

In both cases, execution does not stop immediately, but is gracefully and safely terminated by IG-XL.

```
Services.Alert.Error($"can't apply 'UtilityBitState' to pin '{pin}'");  
  
// ERROR: can't apply 'UtilityBitState' to pin 'AnaIn' [Apply]
```

Where needed, the type of the exception thrown can be specified, to assist users in their troubleshooting:

```
Services.Alert.Error<ArgumentException>("index can not be negative");  
  
// ERROR - RegisterCollection: 'index' can not be negative [this[]]
```

Error alerts are labeled with **ERROR:** and have the calling method listed at the end. Red color / bold style font is used in the OutputWindow.

IMPORTANT

For improved readability and a consistent look and feel, enclose any mentioned objects, variables, values, ... in single quotes ':'

```
Services.Alert.Error($"can't apply 'UtilityBitState' to pin '{pin}'");  
Services.Alert.Warning($"calibration file '{calFile}' not found - using defaults");
```

Since this is free text, the use of ' single quotes can't be enforced by the method, so it'll depend on code authors to make sure this is followed.

Context for Info / Warning / Error Alerts

AlertService uses the `CallerMemberName` feature to get context of the calling method without requiring the user to manually specify (and maintain!) the source. It works with an optional argument decorated with the `[CallerMemberName]` attribute, which the compiler automatically fills out on the caller side. Compared to reflection / stack exploration, this feature works very robust and is essentially free of overhead.

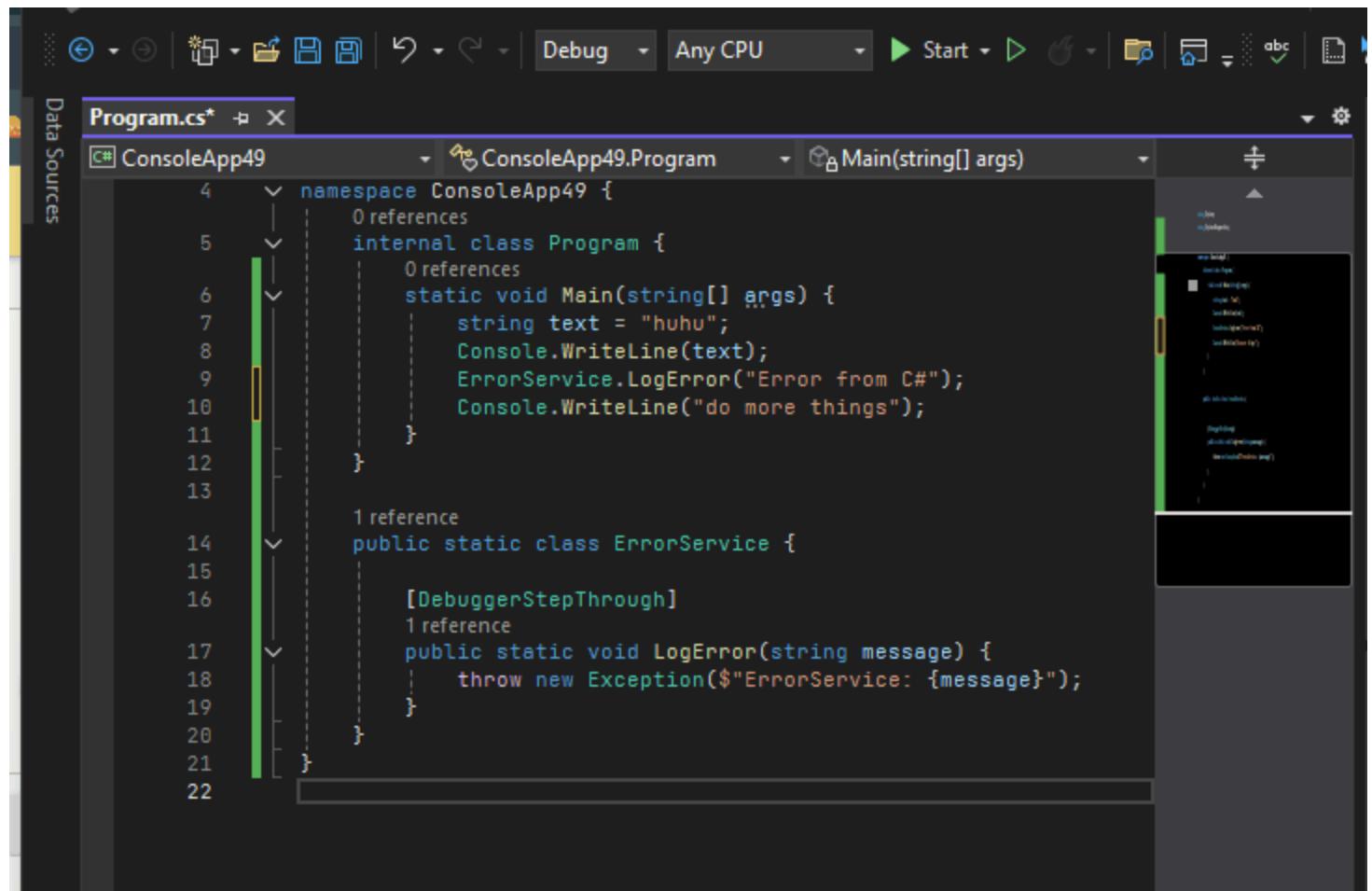
```
/// <summary>  
/// Sends an Info Alert message to the selected output target(s). Use for positive / neutral  
/// information relevant to the user.  
/// </summary>  
/// <param name="info">The Info Alert message.</param>  
/// <param name="doNotSpecify">DO NOT SPECIFY - the name of the calling method is  
/// automatically inserted by the compiler.</param>  
internal static void Info(string info, [CallerMemberName] string doNotSpecify = "") {  
    LogMessage(_infoTarget, GetMessage("INFO", info, doNotSpecify, false),  
    (ColorConstants)0x7f0000, false);  
}
```

WARNING

Users will need to understand they should not manually specify this information. In that case, the compiler feature would be overwritten - no big harm, but the feature is then bypassed. To avoid that, the argument is called `doNotSpecify` and a clear XML API doc info is added.

Interactive Debug

In interactive debug scenarios, code execution is trapped at the offending statement. That way the context like local variables can be analyzed, the execution pointer dragged, EnC changes applied and execution continued:



i NOTE

This feature is pending confirmation within IG-XL. This behavior was captured in a .NET console application. The same experiment in IG-XL test code resulted in an "Unable to set the next statement. The attempt to unwind the callstack failed." error.

Log Alerts

Log alerts are intended for rating-free output of data dumps or outputs for debug or troubleshooting purposes. The message is sent as-is, without any labels or formatting applied.

```
Services.Alert.Log(${myMeasurement});  
  
// outA: 1.225, 1.199, 1.203, 1,212
```

Logs to the OutputWindow can optionally be RGB color formatted and optionally use a bold style. Only in monospaced environments (text datalog window, file), text location formatting / ASCII art can be used, the OutputWindow does not support that with the proportional font that can't be changed.

Output Targets

Three possible output targets are available for the user, where simultaneous outputs to multiple types is supported per user selection. Only the default (= minimum) outputs can not be de-selected.

Alert Type	OutputWindow	Text/Stdf Datalog	File
Log	always	<input type="checkbox"/> optional	<input type="checkbox"/> optional
Info	always	<input type="checkbox"/> optional	<input type="checkbox"/> optional
Warning	always	always	<input type="checkbox"/> optional
Error	<input type="checkbox"/> optional (already logged by IG-XL)	<input type="checkbox"/> optional (already logged by IG-XL)	<input type="checkbox"/> optional

IMPORTANT

For unhandled exceptions, IG-XL will always write message including calling context to the Output Window and the Text Datalog. This behavior is independent from the AlertService and can not be changed. It would therefore be redundant if the AlertService wrote the same information again. Consequently these two output targets are disabled for errors by default.

This also avoids error logs to be shown even in the case of exceptions handled in a `try ... catch` clause. Users may enable these output targets if they wish additional information.

NOTE

The service does not offer a way to globally suppress logs - the sender should instead take care about that. This design choice was made on purpose to avoid test time hits by having the logger perform potentially costly tasks to create an alert (collect data, compose output string) if it is never shown, and therefore goes unnoticed. Supporting a "disable all" could invite hidden overhead and unnoticed, but important alerts.

Time Stamp Headers

Per system-wide setting, a time-stamp can be added in front of Info / Warning / Error Alerts. Can be overridden for individual alerts, off by default.

Validation Fails

A dedicated approach is offered to flag fails of custom validation features to IG-XL to the system. While technically, raising an exception would also prevent validation to succeed, the IG-XL supported API `TheExec.DataManager.WriteTemplateArgumentError` allows collecting multiple validation fails and linking them to their sources. When `AlertService.Error()` is called from within validation, it will automatically use that API internally:

```
[TestMethod, CustomValidation]
public void TestMethod1(int posValue, double negValue) {

    if (TheExec.Flow.IsValidating) {
        if (posValue < 0) Services.Alert.Error("value must be positive", 1);
        if (negValue >= 0) Services.Alert.Error("value must be negative", 2);
        if(someOtherCondition) Services.Alert.Error("some other problem");
        return;
    }

    // some test code
}
```

IG-XL will collectively display these on the Errors sheet:

Errors			
Sheet Name	Cell	Error Code	Error Message
Flow Table (Basic),TM2	AX7	C#RA	some other problem
Flow Table (Basic),TM2	AY7	C#RA	Argument 'posValue': value must be positive
Flow Table (Basic),TM2	AZ7	C#RA	Argument 'negValue': value must be negative

Validation is marked unsuccessful and the offered links lead directly to the offending test method arguments:

Test Procedure		Arg0	Arg1	Arg2	Arg3	Arg4
Type	Name					
.NET	ACME123_NET.TestClass1.TestMethod1	1	-1	1		
.NET	ACME123_NET.TestClass1.TestMethod1					

The one-based argument index is used to highlight the correct input cell on the TestInstances / FlowBasic sheet. If not provided (defaults to `0`), like when the validation check isn't tied to a single argument, the argument list cell (`Arg0`) is marked.

Validation fails are only reported to the IG-XL Validation System, but not logged anywhere else. Code execution will continue through the test of the validation sequence to allow collecting all validation errors.

Alternatives Considered

For a more consistent log it was considered to require `context` specified for every Error / Warning / Info through a non-optional parameter. Users could specify the service, test block, test method ... to help identify the error source. Practically however, inconsistencies increased with some ambiguity specified in the message vs. in that context parameter. Since the `[CallerMemberName]` is available, it was decided that this (automatic) mechanism provides sufficient and reliable context information to alert messages.

Behavior Service

Test programs often reflect a wide range of expectations, habits, and preferences — varying not only between teams, but sometimes even between individuals. What feels "right" for one group might be suboptimal or even undesirable for another. There isn't always a single "correct" or "preferred" way — sometimes, the only solution is to support multiple flavors.

Such behaviors typically have a global scope: they remain consistent throughout the lifetime of a test program. They should be configurable from the outside, without requiring changes to source code — both to protect against unintended side effects and to comply with qualification and auditing standards. Think of a behavior selection as a kind of "Run Mode Option" or "User Preference" — chosen once, respected throughout, and changed only when explicitly stated.

[A list of all implemented **BehaviorService** features can be found here.](#)

IG-XL Options

IG-XL does not natively provide a single, central & common behavior chooser model. Settings - even system-wide - are distributed in multiple different places:

- language APIs (`PinListData.GlobalSort`, ...)
- assembly attributes (`[assembly: RunDspGenerateLegacy]`, ...)
- turd files (`*.dlex`, ...)
- RunOptions (DoAll, ...)
- IG-XL Settings (Waveform Display defaults, ...)
- Oasis (Offline Response Engine, ...)
- EnableWords (for custom, runtime only settings)
- ...

None of these options provide a satisfying model for C#RA specific behavior selection, as they are either not user-extensible or sufficiently capable for the requirements.

Use Cases

As the C#RA and its user base grows, requests for alternative use models are voiced. To date, the following have been captured:

- Parametric Datalogging: use offline readings from drivers vs. send always passing (halfway between limits) values
- FlowLimits vs. LimitSets
- PortBridge vs. no PortBridge
- the use of PatternTags
- AuditMode for SetupService

- Profiling data collection
- Telemetry data collection

It is expected that the BehaviorService can help in all of these cases. In some areas however, additional functionality may be required, specifically when there's design-time dependencies that need to be respected, like the conditional referencing for external libraries like PortBridge.

Use model

As all the other Services in C#RA BehaviorService is centrally available for all test code to interact with stored information. To facilitate Behavior selection the service allows specific **values** to be stored, updated and retrieved for arbitrary **features**. These features can be considered the keys in a dictionary, which supports a strictly typed use model for an exclusive (but extensible) list of data types.

Coupling / Dependencies & Defaults

The proposed use model limits coupling to a minimum and avoids dependencies by using string based keys for the features.

```
Services.Behavior.SetFeature("MaxRetries", 5);
Services.Behavior.SetFeature("Timeout", 30.0);
Services.Behavior.SetFeature("EnableLogging", true);
Services.Behavior.SetFeature("ApiEndpoint", "https://api.example.com");
```

NOTE

Using the `.SetFeature()` method will add that feature entry if it didn't exist, or overwrite the previously stored value. To find out if a certain feature has already been defined, use the `IEnumerable Features` property, which allows to access the entire collection of keys with LINQ methods like `.Contains()`, `.Count()` and more.

All clients of the BehaviorService must implement the concept of a preferred default, which is used if a feature is not defined. The service will in that case return the default for the data type, which must be correctly handled.

```
Console.WriteLine($"max retries: {Services.Behavior.GetFeature<int>("MaxRetries")}");
Console.WriteLine($"hallway lights on: {Services.Behavior.GetFeature<bool>("HallwayLightsOn")}");

// max retries: 5
// hallway lights on: False
```

Data Types

The following data types are supported. Extension is possible but the list is on purpose limited to an exclusive set for robustness and a simple use model:

- `int`
- `double`
- `bool`
- `string`

The BehaviorService is strictly typed, and will retain the original data type for storage and retrieval. Attempts to store a non-supported type, or to read back into a different type than what was used for storage will result in an exception.

File Export / Import

For test program external definitions, the BehaviorService supports file import and export capability using the following syntax:

- one line per **feature**, format `<feature>|<type>|<value>`
- **feature** names can be arbitrary strings, including white space, but not the character `|`
- **type** is the system type name (not the language type)
- **value** can be any content, including empty or `|` characters, but obviously no line breaks
- **value** is parsed to the target type, an exception is thrown in case that fails
- string **values** can't be `null` and are trimmed on storage and retrieval

```
MaxRetries|Int32|5
Timeout|Double|30
EnableLogging|Boolean|True
ApiEndpoint|String|https://api.example.com
```

File import works incremental, thus only overwrites (sets) the features found. To make sure previous definitions are cleared, use the `.Reset()` method.

```
Services.Behavior.FilePath = "c:\\temp\\behavior.txt";
Services.Behavior.Export();
Services.Behavior.Reset();
Services.Behavior.Import();
```

Feature Naming & Scoping

Although this can't be enforced by the SetupService, it's strongly recommended to use a wise naming scheme for the features. With a global scope, special attention should be given to avoid ambiguity or

context issues for users. Features named `TimeOut` or `Logging` are very likely too generic and not indicative enough for users what specific behavior is actually controlled (namespace pollution).

To avoid that, a dot notation hierarchy is proposed, with the domain at the beginning and more specifics separated with `.` characters:

- `ParametricDatalog.Limits: FlowLimits / LimitSets` (type `string`)
- `ParametricDatalog.OfflinePassValues: false / true` (type `bool`)
- `Services.Setup.AuditMode: false / true` (type `bool`)

Alternatives Considered

Standard File Format

The use of XML or JSON file formats was considered for the import / export capability. However, these don't (natively) support generics with type information, so that data readback can be done in a specifically typed way. Instead, everything would be `object` or `string`. Attempting to determine the type from the data would be ambiguous for like `3` which would be interpreted as `string`, `int` or `double`. Specifying the type could be done in JSON or XML, but would then require custom parsers, neglecting the benefits of a standard file format.

Support Custom Defaults for the Features

A generic **BehaviorService** can't both be independent from the client's use cases AND support feature specific defaults at the same time. The avoidance of close coupling is considered critical given how C#RA is planned to extend into areas like RF, optional library extensions (PortBridge), SSN, ...

The defaults are identical to the type's default. Changing that globally, for instance to support the case of pattern threading from `false` to `true` might address the desire for one, but cause headaches for others. Carefully estimating, 50% of the users would be unhappy regardless ...

There are two ways to solve this, probably more after careful consideration. Specifically in this example - the idea is generic - the following could be done:

- invert the logic - call the behavior feature `SharedPatgen`
- use the `string` type with speaking titles for either option and freely define how `""` is interpreted

MethodHandle Type

Overview

`MethodHandle<T>` enables dynamic, type-safe resolution and invocation of static delegates by specifying a `FullyQualifiedName`. Only static methods marked with the `[MethodHandleTarget]` attribute are considered for binding. The delegate is resolved at runtime, adapting to the current execution context (such as debug or production mode).

- `T`: The delegate type. Must inherit from `Delegate`.

Properties

`string FullyQualifiedName`

Gets the fully qualified name of the method to bind to the delegate.

`T Execute`

Gets the resolved delegate instance based on the current execution context.

- If the application is running in **debug mode**, it uses the `_outOfProcessDelegate`.
- If in **production mode**, it uses the `_inProcessDelegate`.

Delegates are lazily initialized and cached per context.

Internal Logic

Execution Context

The execution context is determined by:

```
private bool _outerProcess => TheExec.Flow.RunOption[OptType.Debug];
```

Delegate Resolution

The delegate is created using reflection:

- The fully qualified name is split into type and method.
- The method is searched in the provided assemblies.
- The method signature must match the delegate's signature.
- If found, a delegate is created using `Delegate.CreateDelegate`.

If the method is not found or mismatched, an error is logged via `AlertService`.

Error Handling

- Logs an error if the fully qualified name is invalid.
- Logs an error if the method cannot be found or does not match the delegate signature.

Example

```
// define a static method
[MethodHandleTarget]
public static int Sum(int a, int b) => a + b;

// search for the method
var sum = new MethodHandle<Func<int, int, int>>("Namespace.Class.Sum").Execute;

// execute the method
var result = sum(1, 2);
```

Notes

- Only static methods are supported.
- The delegate is resolved from the calling and executing assemblies.
- Delegates are not serialized; they are re-resolved after deserialization.

Offline Features

Tester availability is regularly a tight asset in the test program engineering phase. Increasing the value of offline setups has a direct and positive impact to the development team's efficiency.

Without actual hardware available, drivers typically rely on a light simulation model. While some hardware registers are cached and programmed values can be correctly read back, measurements and readings that depend on device stimulus usually return defaults. These values may have an undesired effect on the test code mechanics and may result in undesired flow paths or binning.

Use Offline Pass Results

Independent from any readback or calculation, parametric datalogs can be forced to pass by ignoring the incoming data and logging a value mid-way between the limits. That's a simple but effective mechanism to make sure the flow executes as expected, but would not have any effect on test method internal calculations. Functional tests are typically pass already as the drivers don't report fails offline.

C#RA uses a [BehaviorService feature](#) to optionally enable this functionality in `TheLib.Datalog.TestParametric()` block:

Feature	<code>Datalog.Parametric.OfflinePassResults</code>
Type	<code>bool</code>
<code>false</code> (default)	send incoming values values to the datalog
<code>true</code>	query IG-XL for limits and send centered pass values to the datalog

NOTE

This capability is offered independently from a future addition of Offline Response Engine (ORE) support. Since test methods are not changed to support this, users have a free choice of the offline model they wish (none / overwrite / ORE).

Offline Response Engine (ORE)

This new feature in IG-XL can be used to inject offline readback data right into the drivers. That way, the test program can remain completely unmodified while still behaving as if it was running on a tester. A record mode can capture and store actual device responses online, and re-play them in subsequent offline sessions. The offline responses is user editable, so that custom adjustments can be made.

The Offline Response Engine will not be available until IG-XL 11.10 - which also constrains how C#RA can provide such functionality to users.

Alternatives Considered

Conditional Offline Code

The fall-back solution for every test engineer has long been dedicated `if (offline) ...` structures, overwriting the driver's defaults in that case. While this could be done at the test method level, the idea was rejected as too invasive and resulting in cluttered code.

Should users wish to use that model in their own custom test methods, they are totally free to do so. The C#RA provided test methods will not.

Specify Offline Values

It was considered to allow specifying the offline values. This idea was rejected, because it would have required modifying the API to support an additional input. The proposed path of assuming values midway between the low- and high limit is robust and does not require dedicated code. More advanced use cases are deferred to the ORE functionality.

PatternInfo Class

The `PatternInfo` class is a record type that stores basic information about how to run a given pattern. `PatternInfo` instances are used as input to `TestLib.Execute.Digital`'s methods.

By default, patterns will not use threading unless the optional `threading` parameter in the constructor is set to true. If threading is set to true, validation may fail if the pattern is not setup properly for threading.

Definition

```
public class PatternInfo {
    public string Name;
    public string TimeDomain;
    public readonly bool ThreadingEnabled;
    PatternInfo(string patternName, bool threading) {...}
}
```

Test Method Level

```
[TestClass]
public class PatternInfoPrototype : TestCodeBase {

    private PatternInfo _patternInfo;

    [TestMethod]
    public void TestMethod(string pattern) {
        _patternInfo = new(pattern, true);
        TestLib.Execute.Digital.RunPattern(_patternInfo);
        Site<bool> patResult = Acquire.Digital.PatternResults();
    }
}
```

Pins Type

The default way to handle pins, pin groups and lists of these (even nested) is by comma separated strings: these come in from the instance sheet into test methods (choice of supported types is limited here), and is consistently consumed by the PublicAPI features and instrument drivers.

Aside from the performance questions for string handling (overrated ... string manipulation has a bad reputation for being slow, but that's typically when lots of incremental changes are done in loops, but single operations on short strings won't break the bank), there's another aspect to it. For a particular setting to reach a dedicated instrument, the following steps need to be performed:

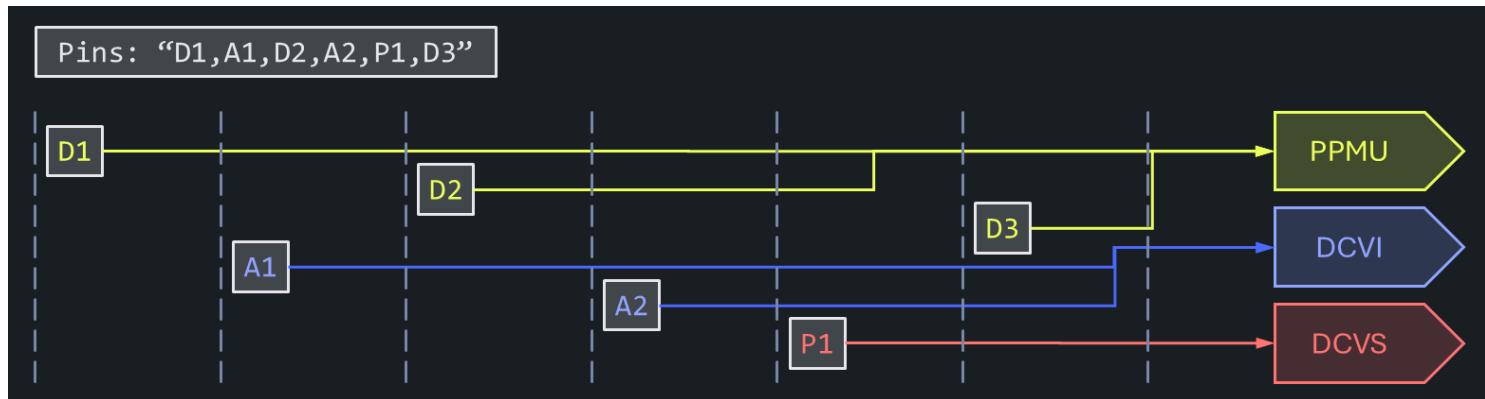
- call PublicAPI driver
- resolve (nested) pin lists, groups and aliases
- identify instrument channels according to site mask
- send data to instruments

IG-XL is pretty capable in handling these steps efficiently. Caching and optimizations under the hood allow broadcasting the data in parallel to all targeted hardware resources.

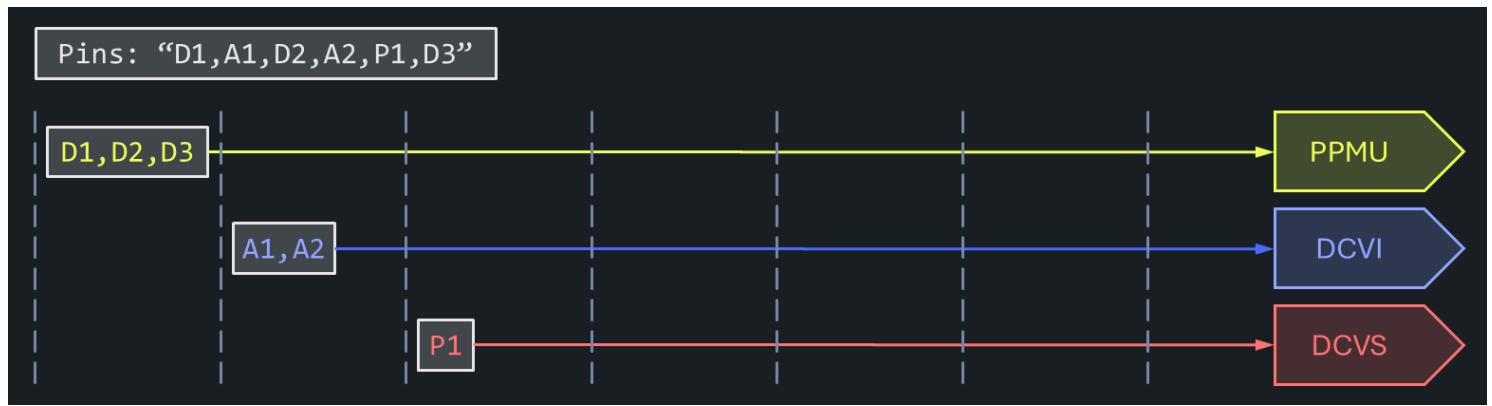
In the case of instrument-agnostic language (goal for our Test Blocks), these additional steps are required **ahead** of the PublicAPI call for efficient operation:

- resolve (nested) pin lists, groups and aliases
- determine instrument type (only for generic code)
- re-group common instrument types into sub-pin-lists

Without that, routing the calls to the specific instruments would have to be done individually. In addition to the (expensive) look-up of the instrument type at runtime, this would result in a complete serial execution:



Caching the instrument types upfront (validation, first-run) allows for optimized combination of alike pins, minimizing the overhead to the bare minimum:



Value-Based Comparison

The `Pins` type supports value-based equality comparison. This is particularly useful when comparing pin configurations across different test scenarios or validating expected vs. actual pin sets.

- **Pin Equality:** Two `Pin` objects are considered equal if their `Name` properties are equal.
- **Pins Equality:** Two `Pins` collections are considered equal if they contain the same sequence of `Pin` objects (by name), in the same order.

This behavior is implemented by overriding `Equals` and `GetHashCode` in both `Pin` and `Pins` to be used reliably in collections like dictionaries or hash sets, and ensures consistent behavior in unit tests and validation logic.

i NOTE

The comparison is **order-sensitive**. That means `[A, B]` is not equal to `[B, A]`.

This design choice aligns with the typical use cases in test development, where the pin order is relevant for calculations or datalog.

Instrument Families

How do we find out which pins can be commonly talked to with the same language statement? Well, that depends ...

Instrument Types

IG-XL itself offers a common API to different (physical) instrument types within the same family. For instance, there's a DC30, DC75, DC80, UltraVI80 for the UltraFLEX and an UltraVI264 for UltraFLEX+ all using the same language node `TheHdw.Dcv...`. Common features that exist on all can easily be accessed with identical calls. Features unique to a specific instrument must still be available on that common node, but will produce a runtime exception if not backed up by the target instrument (like a certain voltage / current range, bleeder resistor, ...).

Instrument Features

Within a physical instrument, multiple (logical) instruments can exist. All digital instruments for instance have a PPMU per channel in addition to the PinElectronic feature. DCVI instruments bring DiffMeter and DCTime features, which all have a separate language node. In some cases this is cross-cutting, so does PPMU features exist on digital instruments, as well those primarily serving mixed signal and RF needs. VI or VS instruments however do **NOT** have PPMUs, as their main purpose already cover this basic capability.

Bottom line - the `Pins` class needs to handle pins that can have a different type, and each contain one or multiple features. Depending on the test code needs, users need to be able to query for either.

Instrument Domain

A core concept of C#RA is the language grouping into functional domains. For instance are all DC related features placed within a `TheLib.Setup.Dc` language branch. That may include PPMU, DCVI or DCVS features.

The `Pins` class can provide information about supported domain(s).

Test Method Level

For an intuitive, light-weight and clear use model. all the complications are buried inside the implementation. Because test instance parameters are limited to very basic types (`string`, `int`, `double` and `bool`), the `Pins` constructor accepts a comma-separated pin list `string` or `PinList` type:

```
[TestClass]
public class PinObjectPrototype : TestCodeBase {

    private Pins _pins;

    [TestMethod]
    public void TestMethod(string pinList) {
        _pins ??= new(pinList);
        TestLib.Setup.Dc.Connect(_pins);
        TestLib.Setup.Dc.ForceV(_pins, 1.0 * V);
        // more test code
    }
}
```

In combination with persistent test class objects, the `Pins` object can be created at first touch only, possibly during validation, so subsequent runs don't see an execution penalty.

Test blocks need to accept the `Pins` type as input. Possibly as an overload in addition to `string` for cases where instrument independent operation isn't needed.

Test Block Level

Inside Test Blocks, `Pins` objects allow convenient querying for the different instrument types and features:

- `ContainsType()`, `ContainsFeature()`, `ContainsDomain()`: does the object contain any pin with the specified type, feature or domain?
- `ExtractByType()`, `ExtractByFeature()`, `ExtractByDomain()`: create a new `Pins` object with only pins of the specified type, feature or domain

This API allows for a concise syntax in generic Test Blocks:

```
public static void Connect(Pins pins) {
    if (pins.ContainsFeature(InstrumentFeature.Ppmu, out string ppmu)) {
        TheHdw.PPMU.Pins(ppmu).Connect();
    }
    if (pins.ContainsFeature(InstrumentFeature.Dcvi, out string dcvi)) {
        TheHdw.DCVI.Pins(dcvi).Connect(t1DCVIConnectWhat.HighForce
| t1DCVIConnectWhat.HighSense);
    }
    if (pins.ContainsFeature(InstrumentFeature.Dcvs, out string dcvs)) {
        TheHdw.DCVS.Pins(dcvs).Connect(t1DCVSConnectWhat.Force | t1DCVSConnectWhat.Sense);
    }
}
```

Merging Results

IG-XL and C#RA heavily rely on pin and site agnostic interfaces. A `TheHdw.Dcvi.Pins().Meter.Read()` will return a single result containing data for all sites and all pins in the specified order. The test block `TheLib.Acquire.Dc.Measure()` does the same. The latter however will have to split the internal action into calls to the dedicated drivers for PPMU, DCVI or DCVS - each returning a separate object.

That object needs to be merged and put into the original sequence requested. The following method allows that:

- `ArrangePinSite<T>()`

Alternatives Considered

The learning cost of a use model for pins that's different from the regular IG-XL way has been carefully considered when selecting the proposed solution.

Option: "Hide" the Caching in a Global Service

Exposing a new use model to Test Method authors could be avoided by creating a global dictionary for all pin list strings that occur in a test program, and having their resolved resources stored in there.

Test Blocks however would need to perform this lookup before processing the individual instruments. Besides the (small) overhead, there's concern about putting this information on a global scope. Rogue Test Blocks could modify this information, potentially impacting other, unrelated areas in a test program. It would be more of a procedural approach and defeat object-oriented principles.

Option: "All-In" OOP Use model

The concept of **Pin** and **Pins** objects could be further enforced by offering methods interacting with the hardware directly on the objects. Implementing dedicated interfaces for instrument types and features could resolve accessing common as well as rather specific aspects.

While appealing from a pure OOP perspective, this approach would require replacing (wrapping) the entire PublicAPI. A fundamental deviation from how things typically work (IG-XL is **NOT** purely OOP), could result in confusion in conjunction PublicAPI. If at all, such a use model should be offered natively by (a future version of) IG-XL.

Implementation

This feature is implemented with gradually added support for instruments as they become required for the respective customer activities.

Coverage

Currently, these instruments are supported:

Instrument (Type)	Feature	Domain	Platform
UP2200	Ppmu, Digital	Dc, Digital	UltraFlex+
UPHP	Ppmu, Digital	Dc, Digital	UltraFlex+ (not tested yet, not in config)
UVS256	Dcvs	Dc	UltraFlex+
UVS64	Dcvs	Dc	UltraFlex+
UVI264	Dcvi	Dc	UltraFlex+
SupportBoard	Utility	Utility	UltraFlex+

Merged Resources

Merged resources both using instrument internal ([DCVSMerged10](#) channel type) or external merging (`:c` syntax) are supported. The [Pins](#) class does not actually know about that, and can not be asked if a resource is merged or not. The IG-XL use model solely requires the leader resource being programmed, and both instrument (base-)type and features are correctly resolved.

Shared Resources

Shared resources both across sites ([siteXX](#) spec in site column) and pins (`S:` syntax) are supported. The [Pins](#) class does not know about this and cannot be asked if a resource is shared.

NC pins

NC return the Type [NC](#) and an empty list of features.

Multiple Resources

Multiple resources connected to a pin using the `:M` syntax is not supported. That concept conflicts with the C#RA principle of instrument agnostic language. For that it is required that a pin has no features with overlapping capabilities. For instance, if a channel map pin is connected to both a Digital and a DCVI instrument, how would a [Setup.Dc.Connect\(\)](#) block know which path to connect?

The `:M` concept cannot be used in C#RA enabled test programs, use dedicated pins instead (like [RST_Dig](#) and [RST_Dcvi](#)).

Search & Trim Tests

Certain device parameters cannot be directly measured and instead require indirect test methods. The test conditions are applied to the device, and the resulting effects - often observed on different pins - are analyzed. The goal is to identify either a tripping point that marks a sudden change in behavior or the input condition that brings the device closest to its ideal performance. Examples include input thresholds, trim parameters, and valid ranges for supply voltages or operating frequencies.

Determining such points typically requires an iterative approach. Even with an ideal tester, the DUT (Device Under Test) itself limits how quickly tests can be executed, making these tests relatively expensive. Extensive searches are often performed during device characterization, while only the most critical ones are retained for production. Once behavior has been thoroughly verified and the production process is stable, these tests are often replaced by faster go/no-go checks. Reducing resolution (i.e., accuracy) is another way to improve test throughput.

Flavors

There is a wide variety of parameters and methods that fall into this category. Below are a few commonly encountered ones.

Functional vs. Parametric

Device behavior may be observed using either functional tests or parametric measurements. While the test approach is largely similar, parametric results are typically pin-based and produce numerical (integer or floating-point) values, whereas functional results are usually binary (pass/fail) patterns.

Search vs. Trim

Trim tests are essentially a specialized form of search. For improved accuracy or performance, some devices allow post-manufacture adjustments via fuse blowing or non-volatile memory programming. A standard search strategy is used to find the optimal trim setting, after which the device is permanently altered.

Characterization & Shmoo

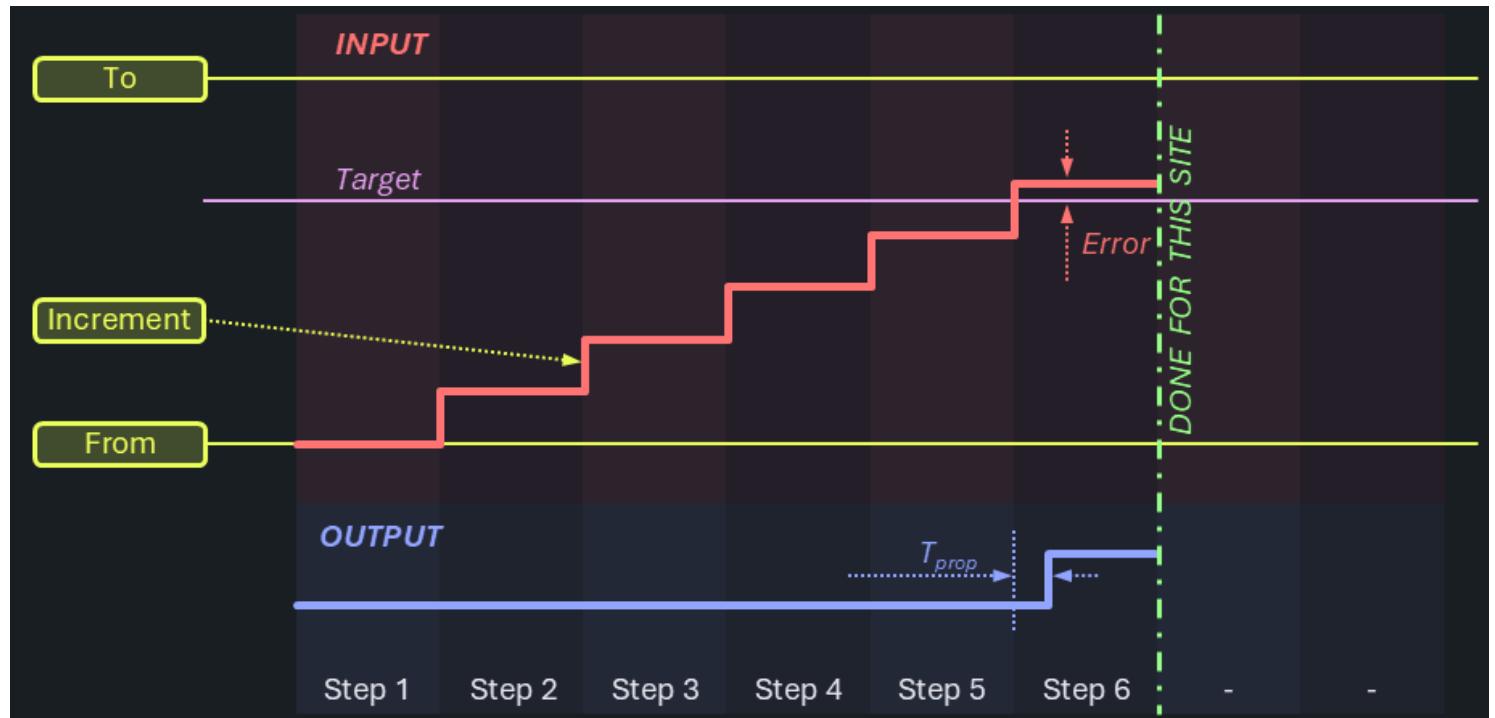
There is a smooth transition between characterization and production testing, as well as between the techniques used. Characterization emphasizes accuracy, often at the cost of speed, while production testing tends to trade off resolution for throughput. Shmoo plots provide a graphical representation of search results, typically in two or three dimensions.

Algorithms

Various search algorithms have emerged depending on DUT characteristics and tester capabilities. Each has its place and may be used interchangeably for comparison or optimization.

LinearStop

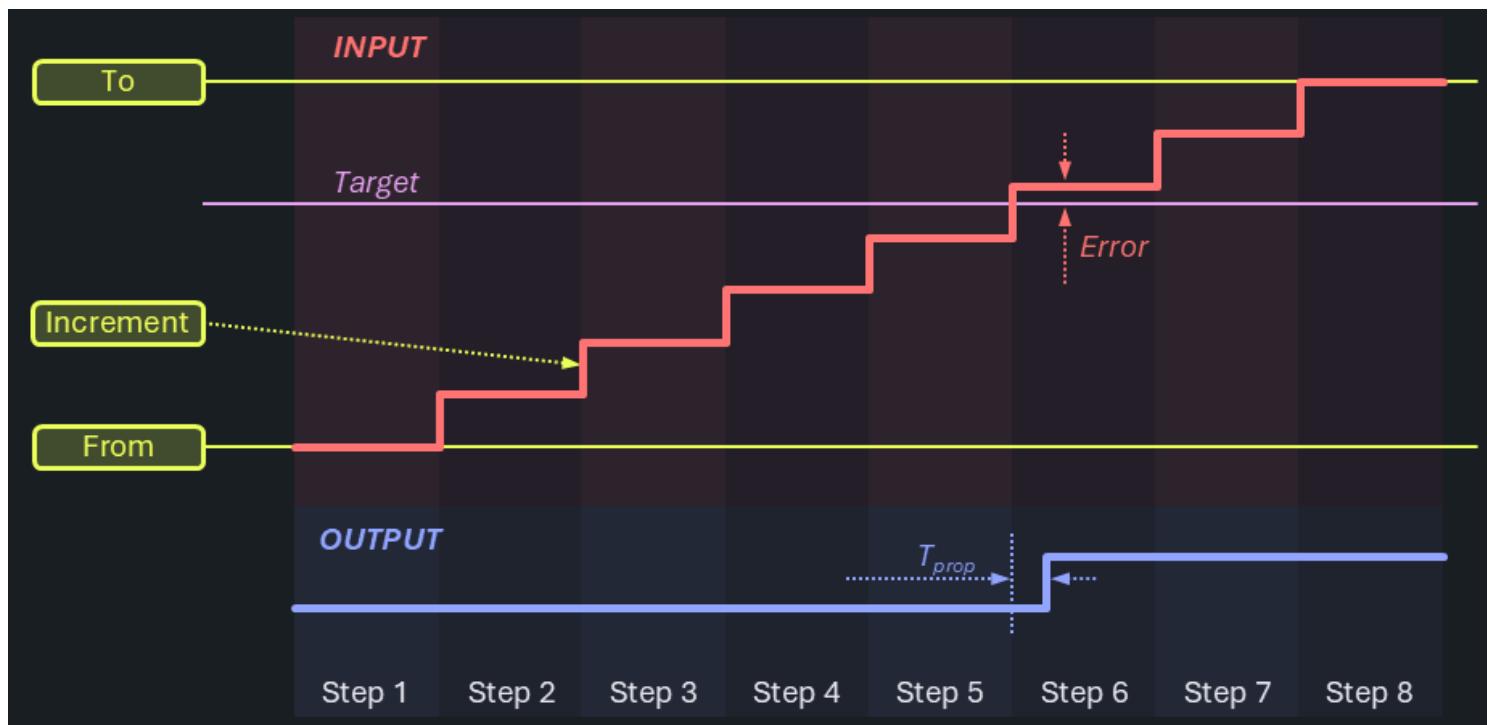
This strategy starts at one end of the range and increments linearly until the desired condition is met. The search is then halted to avoid unnecessary steps. While simple, this approach can be slow and inefficient - especially in multi-site setups - since the full range must be traversed even when the result lies near the opposite end. It works well when the number of steps is small or when device behavior is highly non-linear or otherwise unsuitable for more advanced strategies. It can also serve as a baseline method during early device evaluation.



Since this is an interactive algorithm, it makes real-time decisions based on measurements at each step. This can introduce stalls in pipelined tester architectures.

LinearFull

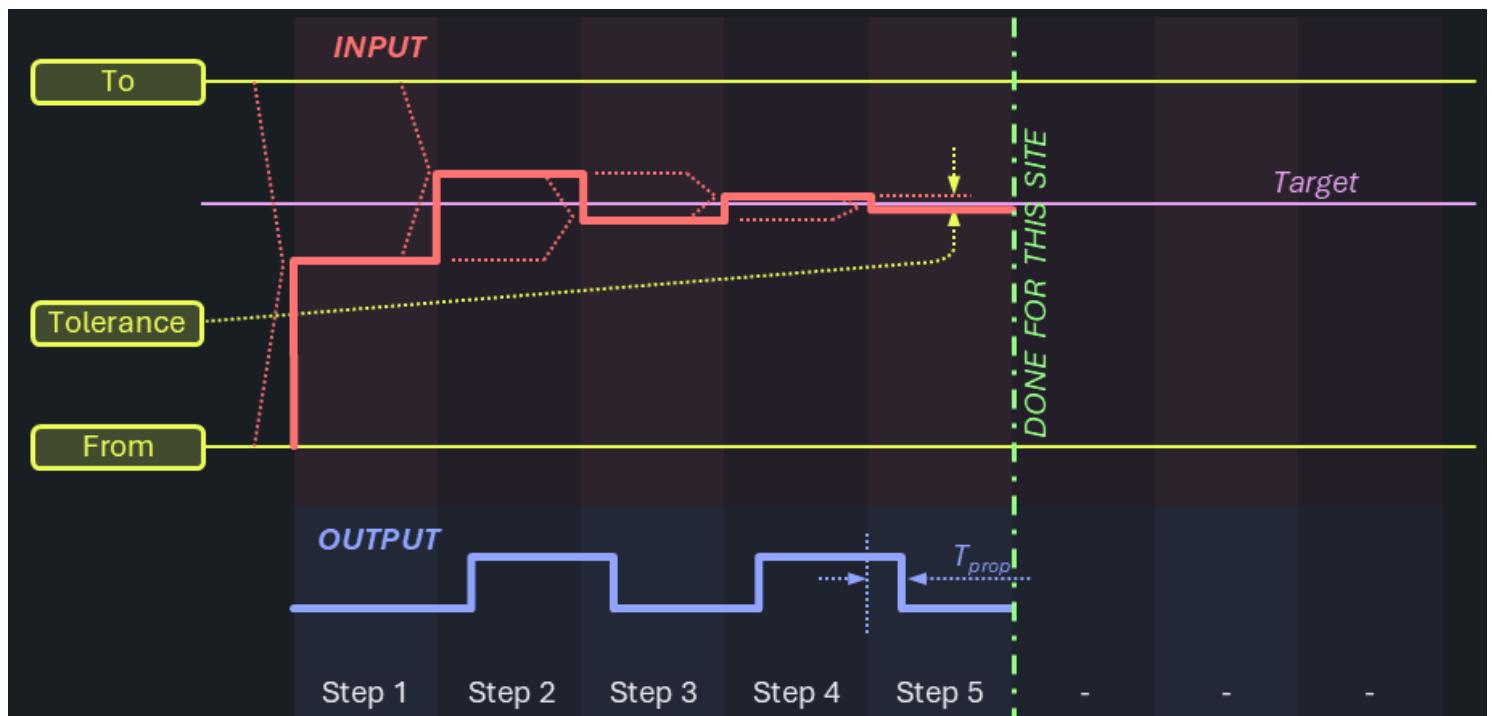
In contrast to LinearStop, this method runs the entire linear ramp regardless of when the condition is met. This can be advantageous for pipelined testers that benefit from deferred measurement processing. If the ramp dynamics are known and consistent, it's possible to read back the full result set efficiently after ramp execution.



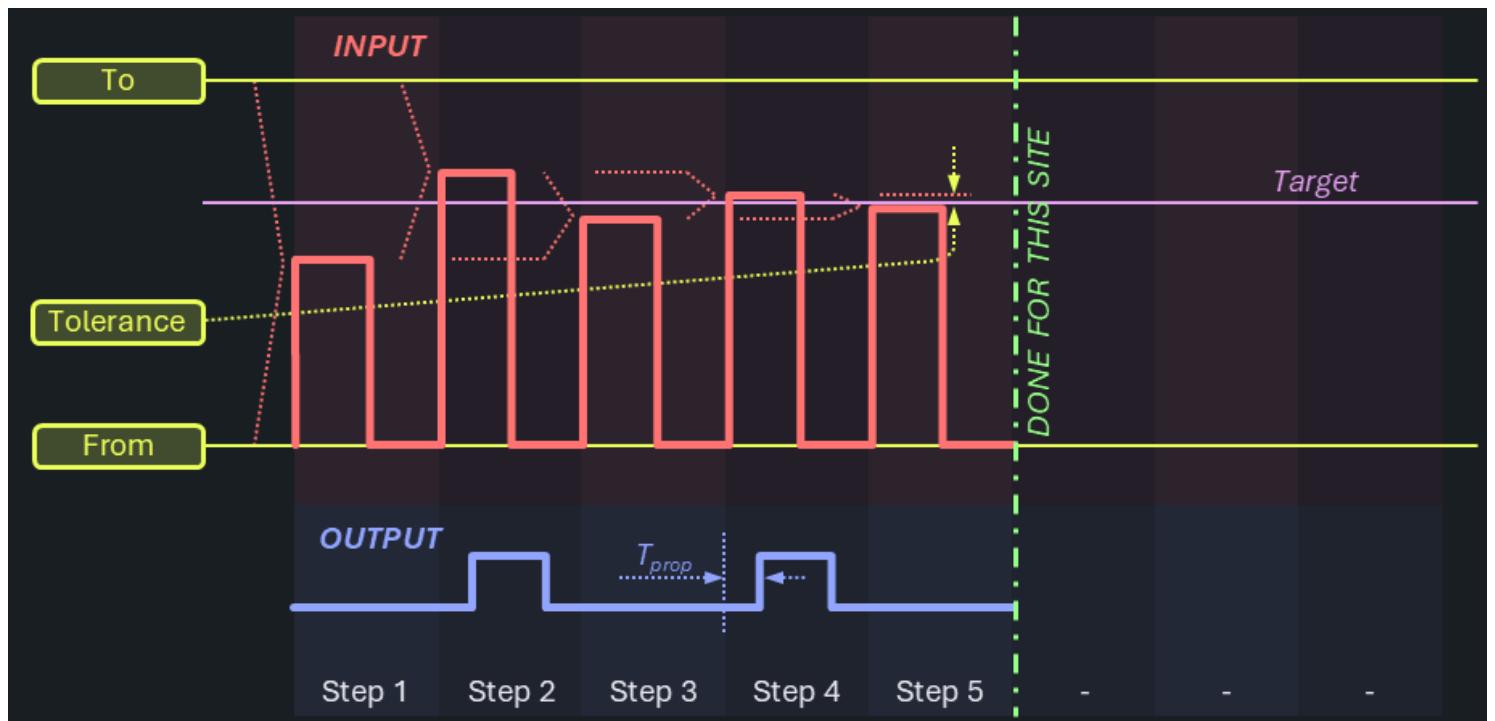
This method is suited to low- to medium-resolution features with manageable DUT settling times. If the settling is slow but predictable, it can be modeled and compensated for.

BinarySearch

The [binary search algorithm](#) significantly reduces the number of steps required - from n to $\log(n)$ compared to a linear search. It is an interactive approach that divides the search space in half at each step, evaluating whether the condition lies above or below the midpoint.



However, this method can be challenging to implement across multiple sites and pins. Each channel requires independent force conditions, limiting the use of broadcast mode. Random noise or hysteresis may reduce repeatability and reproducibility.



Devices with [hysteresis](#) may exhibit bi-modal outcomes due to the search approaching from inconsistent directions. This can be avoided by enforcing a unidirectional search.

Polynomial and Curve Fitting Methods

When a device's transfer function is known or can be reliably modeled, a few strategically selected measurements may suffice to estimate the entire response curve.

- **Exact Polynomial Interpolation** - this method constructs a polynomial of degree $n-1$ that passes exactly through n known data points. It guarantees that all measured points are precisely matched. However, this precision comes at a cost: the resulting curve can become unstable or oscillatory, especially when using high-degree polynomials or noisy data (a phenomenon known as Runge's phenomenon). This approach is best used when the number of data points is small and measurements are highly reliable.
- **Best-Fit Curve Approximation** - instead of requiring the curve to pass through all data points exactly, this method fits a chosen function type (e.g., linear, quadratic, exponential) to the data using regression techniques. The result is an approximate model that minimizes the overall deviation from the measured values (e.g., via least squares). While not exact at any given point, this approach is generally more stable and tolerant of noise, making it well suited for gradual, continuous behaviors.

These modeling techniques are effective in applications such as trim tests, where the device response is smooth and predictable. They are not suitable for discontinuous behaviors like functional tests or

threshold searches, where abrupt transitions cannot be accurately represented by interpolation or smooth curve fitting.

Exotic Algorithms

Some devices limit access to internal features or expose other constraints that require more specialized search strategies. When compatible with the overall design pattern, these can be integrated using C#RA's customization features.

History Based Start Points

If the process is stable and well understood, starting a search from the extremes may be inefficient. Instead, the previous result(s) can serve as a statistical predictor for the current DUT. Searching in alternating directions from this point with increasing resolution can reduce total steps. However, this approach requires persistent data handling beyond a single test program run and individualized force conditions per site/pin.

Coarse / Fine

High-resolution ramps can be divided into coarse and fine phases. The coarse step identifies the general region, while the fine step zeroes in with greater accuracy. This is essentially a hybrid between linear and binary search strategies, where each step improves resolution by a factor (typically two).

Continuous Ramp

Inspired by converter testing at larger resolutions, sticking with a discrete ramp (direct association of every individual step) may become inefficient. Instead, the ramp may be provided as a truly continuous input signal. If the device's detection logic is non-discrete, its response can continuously observed and used as a trigger to detect the source condition, or captured for post-processing. Propagation delay errors - proportional to ramp speed - can be modeled and compensated.

FastRamp With Individual Verify

To compensate for potential inaccuracies from faster ramps (discrete or continuous), a few steps before and after the detected point can be retested for confirmation. If the threshold cannot be verified within tolerance, the device may be binned out.

BinLin / LinBin

BinLin is very similar to the Course / Fine strategy. The Course region typically conducts a binary search. Since things like hysteresis create improper convergence and inconsistencies across sites, it's only used to get within range. From there, the Fine region is conducted using a linear search with much fewer steps than the LinearStop or LinearFull algorithm.

It's currently unclear how LinBin would actually be used.

Built-In Search & Trim tests

Devices may have built-in methods to detect and determine such parameters. What they need to execute that may vary significantly and range from applying precise reference voltage from external to loading executable code into the chip memory and execute pattern to run the algorithm and capture the results.

Those tests are out of scope for this task, as they typically don't use the logic provided here. Instead, they typically use standard features available via different test blocks.

Asynchronous Trigger

Certain instruments allow using a hardware trigger input to determine the stop condition for a signal sourced, like DCVI with DCTime, HVD with EventStamper. To find an input threshold value in that mode, a ramp is sourced to the DUT input, while the output is connected so that a condition change either stops the source signal or records the current force value. This technique can run fully autonomously in the hardware, and the results are read back as the last forcing value from the instruments, specific to multiple sites and pins.

Guidelines

The following guidelines have been captured in the design phase and followed during implementation:

- different search algorithms share common use model and share parameter names for same (similar) features
- execution performance is a critical goal
- functional & parametric tests share common implementation
- user interfaces are not over-burdened with superset of parameters only relevant for a single algorithm
- the risk of over-engineering is mitigated
- C# / .NET type safety is respected
- various input & output types are supported (int, double, bool and going forward - as IG-XL adds support - uint, long, ulong, ...)

Code Structure

Consistent with the core principle of the C#RA, instrument agnostic abstraction language is offered that allows flexible re-use for the common cases and easy extension where custom aspects need to be considered. Due to the broad nature of search & trim tests, the focus is at the test block level, to be used to compose test methods suitable for a specific test. The test methods offered and implemented on the demo device follow that strategy, they may be directly reused where they fit or serve as a blueprint if different requirements exist.

The choice of this architecture was only made after controversial discussions. Eventually, it was found that the benefits outweigh the following downsides:

- algorithm encoded in test blocks, so test methods will have to select one
- test methods provided and shipped with C#RA are likely not "ready-to-use" for large majority of applications
- instead, they serve as template code for "roll-your-own"
- code uses advanced C# language concepts like delegates and generics.

Example Test Case: Parametric Threshold Search

The following test method will determine the input threshold at which the output of the device switches. It'll perform a full ramp where the device's response is captured in every step:

```
[TestMethod, Steppable, CustomValidation]
public void LinearFull_ReadEach(PinList inputPins, PinList outputPins, double voltageFrom,
double voltageTo, int stepCount, double outputThreshold, double stepSettleTime, string setup
= "") {

    if (TheExec.Flow.IsValidating) {
        _inputPins ??= new Pins(inputPins);
        _outputPins ??= new Pins(outputPins);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply();
        Services.Setup.Apply(setup);
        TheLib.Setup.Dc.Connect(_inputPins, true);
        TheLib.Setup.Dc.Connect(_outputPins, true);
    }

    if (ShouldRunBody) {
        TheLib.Setup.Dc.ForceV(_inputPins, voltageFrom);
        TheLib.Setup.Dc.ForceHiZ(_outputPins);
        double increment = TheLibX.Acquire.Search.LinearFullFromToSteps<double>(voltageFrom,
voltageTo, stepCount, ForceAndRead);
        _tripVoltage = TheLibX.Acquire.Search.LinearFullProcess(measurements, voltageFrom,
increment, 0, -999, m => m > outputThreshold);
    }

    if (ShouldRunPostBody) {
        TheLib.Setup.Dc.Disconnect(_inputPins, false);
        TheLib.Setup.Dc.Disconnect(_outputPins, false);
        TheLib.Datalog.TestParametric(_tripVoltage, voltageFrom);
    }
}
```

```

    void ForceAndRead(double inValue) {
        TheLib.Setup.Dc.ForceV(_inputPins, inValue);
        TheHdw.SettleWait(stepSettleTime);
        measurements.Add(TheLib.Acquire.Dc.Measure(_outputPins));
    }
}

```

The `Validating`, `PreBody` and `PostBody` sections perform the normal and typical data preparation, setup, reset and datalog actions - they are not further explained here.

The `Body` section starts by preparing the instrument setups on the device's inputs and outputs. A data container `List<PinSite<double>> measurements` is prepared to hold each step's output. Then, the test block `TheLib.Acquire.Search.LinearFullSweepFromTo` performs the complete ramp sweep according to the input parameters.

The last parameter is a delegate to the method to be performed for every, which only takes the current force value as an input parameter.

Local methods have access to all local variables and class fields, which is a great benefit. The `_inputPins`, `stepSettleTime` and `measurement` container for the results don't have to be passed in as arguments, but can be directly used. This makes the concept very generic, as it avoids impacting the argument signature of the test block. That can follow a flexible design with the ramp control parameters being generic:

```

internal static Tin LinearFullFromToSteps<Tin>(Tin from, Tin to, int steps, Action<Tin>
oneStep) {
    dynamic inValue = from;
    dynamic increment = (to - inValue) / (steps + 1);
    for (int i = 0; i < steps; i++) {
        oneStep(inValue);
        inValue += increment;
    }
    return increment;
}

```

Overloads are offered for alternative ramp definition models to take calculations off of users shoulders:

```

public static void LinearFullFromToInc<Tin>(Tin from, Tin to, Tin increment, Action<Tin>
oneStep) { .. }

public static void LinearFullFromIncSteps<Tin>(Tin from, Tin increment, int steps,
Action<Tin> oneStep) { .. }

```

Finally, the collected measurements are processed to find the tripping point. This is done by the `TheLib.Acquire.Search.LinearFullProcess` test block, executed as a separate step after the ramp has completed. It uses a `Func<double, bool>` delegate to describe the criteria, in this case when the `outputThreshold` is exceeded.

```
public static PinSite<Tin> LinearFullProcess<Tin, Tout>(List<PinSite<Tout>> measurements,
Tin from, Tin increment, Tin offset, Tin notFoundResult, Func<Tout, bool> tripCriteria) {
    PinSite<int> tripIndex = measurements.First().Select(ps => -1); // Clone the structure
of the first measurement
    ForEachSite(site => { // this could be done with a convoluted & unreadable
LINQ statement
        for (int index = 0; index < measurements.Count; index++) {
            for (int pin = 0; pin < tripIndex.Count; pin++) {
                if (tripIndex[pin][site] == -1 && tripCriteria(measurements[index][pin]
[site])) tripIndex[pin][site] = index;
            }
        }
    });
    return tripIndex.Select(t => t > -1 ? (Tin)(t * (dynamic)increment + from + offset)
: notFoundResult);
}
```

Lambda expressions

Instead of a named method, lambda expressions (`inValue => ...`) may be used to pass in the code to be executed per step, following the same notation IG-XL proposed for `ForEachSite()` loops.

```
List<PinSite<double>> measurements = new();

//if (TheExec.Flow.IsValidating) { ... }      //if (ShouldRunPreBody) { ... }

if (ShouldRunBody) {
    TheLib.Setup.Dc.ForceV(_inputPins, voltageFrom);
    TheLib.Setup.Dc.ForceHiZ(_outputPins);
    List<PinSite<double>> measurements = new();
    double increment = TheLibX.Acquire.Search.LinearFullFromToSteps<double>(voltageFrom,
voltageTo, stepCount, inValue => {
        TheLib.Setup.Dc.ForceV(_inputPins, inValue);
        TheHdw.SettleWait(stepSettleTime);
        measurements.Add(TheLib.Acquire.Dc.Measure(_outputPins));
    });
    _tripVoltage = TheLibX.Acquire.Search.LinearFullProcess(measurements, voltageFrom,
increment, 0, -999, trip => trip > outputThreshold);
}
```

```
//if (ShouldRunPostBody) { ... }
```

This notation may be preferable for cases where the delegate code is not too complex or convoluted. There is a little learning curve with the lambda expressions syntax (`=>`), so that teams at the beginning of the learning curve with C# may prefer explicit methods instead.

NOTE

There is no functional or performance difference between lambda expressions or named local methods. The compiler will actually turn this into the exact same IL code.

Public Methods

Only where non-local methods are desired, the local context needs to be handed through parameters:

```
if (ShouldRunBody) {
    TheLib.Setup.Dc.ForceV(_inputPins, voltageFrom);
    TheLib.Setup.Dc.ForceHiZ(_outputPins);
    List<PinSite<double>> measurements = new();
    double increment = TheLib.Acquire.Search.LinearFullSweepFromTo<double>(voltageFrom,
    voltageTo, stepCount, inValue =>
        CentralMethodsSomewhereElse.OneStepReadCollect(_inputPins, _outputPins, inValue,
        stepSettleTime, measurements));
    _tripVoltage = TheLib.Acquire.Search.LinearFullProcess(measurements, voltageFrom,
    increment, 0, -999, m => m > outputThreshold);
}
```

Read All Results In Once

In this example the measurements per step are only strobed, and then read back in a single shot after the ramp completes:

```
if (ShouldRunBody) {
    TheLib.Setup.Dc.ForceV(_inputPins, voltageFrom);
    TheLib.Setup.Dc.ForceHiZ(_outputPins);
    double increment = TheLibX.Acquire.Search.LinearFullFromToSteps<double>(voltageFrom,
    voltageTo, stepCount, ForceAndStrobe);
    PinSite<Samples<double>> measurements =
    TheLib.Acquire.Dc.ReadMeasuredSamples(_outputPins, stepCount);
    _tripVoltage = TheLibX.Acquire.Search.LinearFullProcess(measurements, voltageFrom,
    increment, 0, -999, trip => trip > outputThreshold);
```

```

}

void ForceAndStrobe(double inValue) {
    TheLib.Setup.Dc.ForceV(_inputPins, inValue);
    TheHdw.SettleWait(stepSettleTime);
    TheLib.Acquire.Dc.Strobe(_outputPins);
}

```

The exact same test blocks can be used to achieve this. The per-step-action performs a `Dc.Strobe` only and a dedicated statement reads the measured samples after completion. An overload is offered for the processing block to support the slightly different result object structure `PinSite<Samples<double>>`. This model is also suitable for embedded DSP processing for best performance and full backgrounding.

LinearFull with Functional Result

```

if (ShouldRunBody) {
    TheLibX.Acquire.Search.LinearFullFromToInc<double>(start, stop, increment, FuncStep);
    _tripVoltage = TheLibX.Acquire.Search.LinearFullProcess(_measurementsPs, start,
increment, 0, -999, trip => trip);
}

void FuncStep(double inValue) {
    TheLib.Setup.Dc.ForceV(_pins, inValue);
    TheLib.Execute.Digital.RunPattern(_pattern);
    _measurementsPs.Add(new PinSite<bool>(TheLib.Acquire.Digital.PatternResults()));
}

```

Functional test results can be handled with `bool` as the output type. Since pattern results are not per pin, `Site<bool>` may be used instead.

LinearStop Algorithm

A different search algorithm can be applied by simply switching the test block to the `TheLib.Acquire.Search.LinearStopFromTo` block:

```

if (ShouldRunBody) {
    TheLib.Setup.Dc.ForceV(_inputPins, voltageFrom);
    TheLib.Setup.Dc.ForceHiZ(_outputPins);
    _tripVoltage = TheLibX.Acquire.Search.LinearStopFromToSteps<double, double>(voltageFrom,
voltageTo, stepCount, 0, -999, ForceAndRead,
    m => m > outputThreshold);
}

PinSite<double> ForceAndRead(double inValue) {

```

```

    TheLib.Setup.Dc.ForceV(_inputPins, inValue);
    TheHdw.SettleWait(stepSettleTime);
    return TheLib.Acquire.Dc.Measure(_outputPins);
}

```

It follows the same use model, but instead of a separate processing step, it directly returns the result. This is a benefit of the interactive search algorithms, however at the cost of stalling the pipeline.

BinarySearch

In the same way, a binary search can be used thought the test block

`TheLib.Acquire.Search.BinarySearch:`

```

if (ShouldRunBody) {
    TheLib.Setup.Dc.ForceV(_inputPins, voltageFrom);
    TheLib.Setup.Dc.ForceHiZ(_outputPins);
    _tripVoltage = TheLibX.Acquire.Search.BinarySearch<double, double>(voltageFrom,
voltageTo, minDelta, false, -999, ForceAndRead,
    m => m > outputThreshold);
}

PinSite<double> ForceAndRead(PinSite<double> inValue) {
    //TheLib.Setup.Dc.ForceV(_inputPins, level); // no multi-pin / multi-site flavor
available yet
    TheHdw.SettleWait(stepSettleTime);
    return TheLib.Acquire.Dc.Measure(_outputPins);
}

```

BinarySearch unidirectional:

To avoid hysteresis problems, a modified binary search concept can make sure to always approach the levels from the same direction:

```

if (ShouldRunBody) {
    TheLib.Setup.Dc.ForceV(_inputPins, voltageFrom);
    TheLib.Setup.Dc.ForceHiZ(_outputPins);
    _tripVoltage = TheLibX.Acquire.Search.BinarySearch<double, double>(voltageFrom,
voltageTo, minDelta, false, -999, ForceAndRead,
    m => m > outputThreshold);
}

PinSite<double> ForceAndRead(PinSite<double> inValue) {
    //TheLib.Setup.Dc.ForceV(_inputPins, level); // no multi-pin / multi-site flavor
available yet

```

```

    TheHdw.SettleWait(stepSettleTime);
    var meas = TheLib.Acquire.Dc.Measure(_outputPins);
    TheLib.Setup.Dc.ForceV(_inputPins, voltageFrom);
    TheHdw.SettleWait(stepSettleTime);
    return meas;
}

```

This is achieved by modifying the single step lambda expression to return back to the starting point after each measurement. The test block itself is agnostic and can be used without change.

Trim Test using LinearFull

Again without modification, the test blocks support trim tests:

```

if (ShouldRunBody) {
    RegisterWrite(from);
    TheLib.Setup.Dc.ForceHiZ(_outputPins);
    TheLibX.Acquire.Search.LinearFullFromToInc<int>(from, to, 1, ForceAndRead);
    _trimCode = TheLibX.Acquire.Search.LinearFullProcess(measurements, from, 1, 0, -999,
trip => trip > target);
}

void ForceAndRead(int inValue) {
    RegisterWrite(inValue);
    TheHdw.SettleWait(stepSettleTime);
    measurements.Add(TheLib.Acquire.Dc.Measure(_outputPins));
}

```

In this case the input type is `int` for the trim code. Instead of a threshold, the trip criteria is specified by the output value exceeding the trim target, but alternative concepts (last step BEFORE switch, minimum absolute deviation, ...) are possible. Where alternative strategies are required (`LinearFull_ReadAll`, `LinearStop`, `BinarySearch`, ...), the respective test blocks can be used instead.

Options

Flexiblitiy is required to suit the various use cases captured for this wide range of test techniques. Test blocks are overloaded for:

- Ramp Definition
 - `from, increment, steps` - specify start point, step size and count
 - `from, to, increment` - specify start point, end point and step size
 - `from, to, steps` - specify start point, end point and step count
- Search Criteria
 - `target` - output value that should be matched closely

- `tripCriteria` - delegate to indicate inflection point
- Results
 - `inValue` - best input for the desired output (method return)
 - `index` - ramp index for identified best input (out parameter)
 - `outValue` - actual output value for identified best input (out parameter)

Alternatives Considered

Search & Trim - A Vibrant Domain

A variety of user preferences, flavors, concepts and styles exist for what is considered "the best" approach to this domain. Despite intense discussions and prototyping, it was difficult to agree on a single concept to move forward with this. Undeniably, all of the proposed alternatives have other benefits and / or solve a particular aspect more elegantly. The proposal above was determined in team voting as the most reasonable compromise to move forward.

The alternative ideas are shown here to allow closing the loop on how well the chosen path turns out to be a viable solution. A implementation proceeds, regular check-steps confirm whether design choices work out. Should the chosen path run into a road-block, alternatives may be revived quickly.

Flat Algorithm in Test Method

Pros:

- Everything is local to the test method
- Easy to read and edit

Cons:

- May be viewed as too simple
- Algorithm would be duplicated in multiple classes
- Any customization requires a new test method

```

if (ShouldRunBody) {
    _measurements.Clear();
    for (double value = start; value <= stop; value += increment) {
        _measurements.Add(FuncStep(value));
        _tripIndex = FindTransition();
        if (!_tripIndex.Any(i => i == -1)) break;
    }
}

if (ShouldRunPostBody) {
    Site<double> tripValues = _tripIndex.Select(t => (t * increment + start));
    TheLib.Datalog.TestParametric(tripValues);
}

```

This is the most straight-forward approach, showing the loop mechanics right in the test method. Users would not have to jump to different places when stepping through, everything would be very easy to follow.

Advanced code concepts like delegates and generics are avoided at the cost of specific implementations for the given task.

Especially for more complex algorithms, this approach could result in complex test methods. There's little encapsulation or reuse applied.

Delegates by Name

```
using System;
[TestMethod, Steppable, CustomValidation]
public void LinearFull(Pattern pattern, string forcePins, double start, double stop, int steps, string stepAction, string tripAction, string setup = "") {
    if (TheExec.Flow.IsValidating) {
        if (!string.IsNullOrEmpty(pattern)) {
            strArgs["pattern"] = pattern;
        }
        strArgs["forcePins"] = forcePins;
        _stepAction = TheLib.Validation.GetDelegate<Func<Dictionary<string, string>, Dictionary<string, double>, Site<bool>>>(stepAction);
        _tripAction = TheLib.Validation.GetDelegate<Func<List<Site<bool>>, Site<int>>>(tripAction);
    }

    if (ShouldRunPreBody) ...

    if (ShouldRunBody) {
        _tripValues = TheLibX.Acquire.Search.LinearFull(start, stop, steps, strArgs, dblArgs, _stepAction, _tripAction);
    }

    if (ShouldRunPostBody) {
        TheLib.Datalog.TestParametric(_tripValues);
    }
}
```

Pros:

- Delegates are tm params
- Delegates can be in any library
- Search functionality can be altered without tm modification
- Algorithm is in test block
- Dictionary content can vary for each tm

Cons:

- Dictionaries lack defined structure
 - Missing elements are runtime errors
- Overhead to recreate objects in static delegates

```
namespace CustomerLib {

    public static class FuncActions {

        public static Site<bool> FuncStep(Dictionary<string, string> strArgs, Dictionary<string, double> dblArgs) {
            Pins forcePins = new Pins(strArgs["forcePins"]);
            PatternInfo pattern = new PatternInfo(strArgs["pattern"], true);
            TheLib Setup.Dc.ForceV(forcePins, dblArgs["stepValue"]);
            TheLib.Execute.Digital.RunPattern(pattern);
            return TheLib.Acquire.Digital.PatternResults();
        }

        public static Site<int> FindTransition(List<Site<bool>> measurements) {
            Site<int> tripIndex = new Site<int>(-1);
            foreach (site => {
                for (int i = 1; i < measurements.Count; i++) {
                    if (measurements[i][site] != measurements[0][site]) {
                        tripIndex[site] = measurements[i][site] ? i : i - 1;
                        break;
                    }
                }
            });
            return tripIndex;
        }
    }
}
```

This approach may be the plan-of-record when truly generic test methods are desired, where users can flexibly select the algorithm as a test method input parameter.

The generalization comes at the cost of some overhead and limitations in typing, which may be acceptable where flexibility is required.

Search by ConditionList

Search Primitives

- A list of conditions to iterate through
 - Fully defined before actual search?
 - Dynamically generated during the search? (improvise)
- An action to be performed upon each condition
 - Capable of flagging the search to end immediately
 - Optionally it can feedback to the algorithm in case the next condition is dynamically generated
- An algorithm which will generate the condition list from very few parameters
 - or generate the next condition with additional feedback from the step action.
- A class named `ConditionList<Tin>`
 - Inherited from `Queue<Site<Tin>>`
 - Conditions are always `Site<T>`
- A function that takes one condition (`Site<Tin>`) and return a flag that can end the search prematurely `Func<Site<Tin>, bool>`
- A function that can generate a `ConditionList<Tin>` object
 - or modify an existing one with additional info (`Site<Tin>`)

Search with Static Conditions

```
// =====
// do a Linear search, condition list generated by RA Algorithm
// =====

TheLibX.Acquire.Search.FromToStatic(
    Algorithm.LinearSteps(voltageFrom, voltageTo, stepCount),
    PerStepTest);

// alternatively create the condition list with new()

TheLibX.Acquire.Search.FromToStatic(
    new ConditionList<double>(voltageFrom, voltageTo, stepCount),
    PerStepTest);

// or use a custom algorithm to generate the condition list

TheLibX.Acquire.Search.FromToStatic(
    CustomExpConditionList(1.0, 10),
    PerStepTest);
```

```
// reusable customer owned code block.
bool PerStepTest(Site<double> voltage) {
    //TheLib.Setup.Dc.ForceV(_inputPins, voltage); // this need to be changed to support Site<double>
    ForEachSite(site => TheLib.Setup.Dc.ForceV(_inputPins, voltage[site]));
    // per step test result are stored elsewhere and decoupled from the search RA
    measurements.Add(TheLib.Acquire.Dc.Measure(_outputPins)); // adding a test history for future analysis.
    // per step test results can be post processed or evaluated immediately and generate a stop condition.
    return false; // true: stop the search
}
```

```
// custom algorithm to generate the condition list, Exponential decay for example:
ConditionList<double> CustomExpConditionList(double from, int count) {
    ConditionList<double> conditionList = new();
    for (int i = 1; i <= count; i++) {
        conditionList.Enqueue(from);
        from /= 2;
    }
    return conditionList;
}
```

```

// =====
// Dynamic search, where conditions are not predictable.
// =====
TheLibX.Acquire.Search.FromToDynamic<double, bool>(new ConditionList<double>(voltageFrom, voltageTo, 10),
    TheLibX.Acquire.Search.Algorithm.BinarySteps,
    (voltage, isNotEnough) => {
    //TheLib.Setup.Dc.ForceV(_inputPins, voltage); // this needs to be changed to support Site<double>
    ForEachSite(site => TheLib.Setup.Dc.ForceV(_inputPins, voltage[site]));
    // per step test result are stored elsewhere and decoupled from the search RA:
    var readings = TheLib.Acquire.Dc.Measure(_outputPins); // perform actual measurement, generating a PinSite<double>
    measurements.Add(readings); // adding to history for further analysis.
    isNotEnough = readings.Max() <= outputThreshold; // this parameter will be returned and used to update the next condition.
    // per step test result can be post processed or evaluated immediately and generate a stop condition.
    return false; // true: stop the search
});

```

Search with Dynamic Conditions

```

public static void FromToDynamic<Tin, Tout>(
    ConditionList<Tin> conditionList,
    Action<ConditionList<Tin>, Site<Tout>> algorithm,
    Func<Site<Tin>, Site<Tout>, bool> perStepRoutine) {
    // loop until conditions are no more generated or a flag is set.
    while (!conditionList.IsEmpty()) {
        Site<Tout> algoParam = new();
        bool finished = perStepRoutine(conditionList.CurrentValue, algoParam);
        if (finished) return;
        algorithm(conditionList, algoParam);
    }
}

```

Features	Pros	Cons
Only 2 interfaces: Static & Dynamic	Simple use model, user do not need to change to a different function if they want to try a different algo	Changed the mind set, instead of saying "I want to search from A to B with step C" now becomes "try each condition in the queue"
Predefined algorithms: <u>LinearFromToSteps</u> , <u>LinearFromToIncrement</u> , <u>Binary</u> , etc.	Easy for a quick start	TBD: how many predefined algorithms should we provide
extend to customized algorithms	Decoupled algorithm from the main search interface now makes it easier for user to focus on the algorithm, and the actual list of performed conditions can be plotted out visually	A little bit hard for users to understand the model and write their first algorithm.
Decoupled from Search result calculation	Makes use model simpler	Not very intuitive to get the search result

What else is possible if we go down this path

- Things like polynomial fit is really an algorithm that process upon a set of test results with known conditions, this is decoupled now and can be later implemented as another layer of result processing
- If we record the test result for each test condition in another class later, (or the same ConditionList class but we need to name it differently) we could do more with that. Such as adding result processing functions like FindFirstPassToFail... this could also be a DSP function.

This approach has the benefit of strictly following a single, generic model for any search. All steps are present, which can help when a search history would need to be traced or plotted.

Some concerns were raised on the resulting code complexity in the test blocks. Even though additional algorithms could be implemented in a very concise way, there was some learning investment for the first. Users only wishing to step through the existing code might find it challenging to understand. The generalization may result in dictionary types being a little intransparent on the meaning of their contents.

The All-In-One Test Method

It's expected that test methods require some modifications / adjustments to the specific device in most cases. The variety of conditions and parameters is just too broad. Offering a variety of specialized test methods templates as starting point for customization is considered better than an overly generic one with significant complexity.

That doesn't get in the way of offering overlapping functionality like with a dedicated functional Vmin test method that allows choosing between linear and binary search, if there are tangible use cases and reasonable functional scope can be determined.

Adjust

TODO

IG-XL's Adjust Service (?) offers similar services and may be used to achieve the same. It's currently not clear if and how that is an option, or if only concepts and ideas should be reused.

Multi-Pin

The original concept envisioned a multi-pin model with single pin being only a special case. It turned out that this would heavily complicate implementation with a very questionable benefit. Independent but simultaneous searches on multiple input --> output paths within the same device are a very rare case.

It was decided to use a single-pin model for searches.

Questions

- POP for linear full?

Setup Service

Much of the test coverage of typical devices can be achieved with only a handful test methods, which are applied with varying parameters. What's typically very specific to a device and test solution however is the required setup to execute such tests.

The **SetupService** provides a generic solution for defining, applying and verifying the correct setup for tester, interface hardware and device. It offers means to manage setups, built-in logic for efficient use as well as auditing and diagnosis capability.

Problem Statement

"Managing tester, interface hardware and device setups is a common challenge for test solutions. It gets in the way of common code, adds overhead to test programs, a significant test time optimization burden and can be the source of hardware reliability problems if not done correctly. The diverse nature of various possible setup aspects makes a generic solution tricky."

Traditional Approaches

The problem isn't new, and so are the various attempts that have been made to address it.

Fully Specify All Setups

A safe path may seem to fully specify all setup requirements for all tests. Aside from bloating the test program, this approach can add significant overhead to execution - even applying the same settings like programming an instrument to the same output voltage, will cost test execution time.

Trying to add parameters to standardized test methods that allow for flexible pre-setup options (like utility bits to switch on & off, digital hold states, preconditioning patterns, instrument connects, ...) quickly becomes overwhelming - too diverse are the options that may exist.

Incremental Setups

Partly alleviated is that problem with test methods that are designed for an incremental approach: in a series of similar tests the correct setup would only have to be applied once and could then be reused. Only the setup part would be device specific, while the actual tests can be generic and avoid any overhead caused by redundant setups.

Unfortunately such an approach makes the test sequence highly susceptible to changes further up in the flow and issues introduced by sequence changes. Problems may show up in completely unrelated areas of the test program, and identifying the correct place for a fix might be tricky.

Another downside is the lack of explicitness. Users unfamiliar with the test program may find it difficult to determine which setup is applied and if it's the correct one.

Overall, this approach defies basic principles of modularity and separation of concerns.

Full Setups for Debug, Incremental for Production

Combining the two may seem like the best of two worlds, with full support for flexibility in a debug scenario and a tailored & efficient solution for production. An `if(!production) { ... }` statement would enclose those settings that are not required in an optimized flow.

As it turns out, maintaining two scenarios often causes increased maintenance effort, which is often given up as production needs tighten. The two different paths through the test programs start to diverge, quickly to a point where the debug option isn't usable anymore.

Default Setups

Agreeing on a common set of conditions that's good for most of the tests can limit the overhead. Once implemented, all tests would assume that setup to be applied as they start. Any deviations they need from that they could apply, but would have to make sure to revert before they end. Violations may not be easy to locate, but once isolated, are straight-forward to fix.

In reality however, agreeing on such a common setup may be difficult depending on the device / interface / setup complexity. Parameters without a clearly dominating state would still require lots of switching, which in case of relays may result in increased hardware wear impacting system reliability.

None of the concepts above suit well for common library based test methods, as they trade-off customization needs with execution performance.

The Smart Way

All the described ideas above try to optimize for a certain scenario at the cost of the other. The **SetupService** resolves that by combining them and using logic to find the most efficient way transparently for the user.



Named Setups

Named **setups** are defined and centrally maintained for a specific test program. To apply a **setup**, it is only required to provide the identifier, and the service will know what do to. Participating test methods can use a single **string** type argument simplifying the idea of commonly usable library functions, as the specific setup aspects are encapsulated in a generic container.

This concept is consistent with how levels and timing context is handled, they also combine a variety of different conditions under a single identifier.

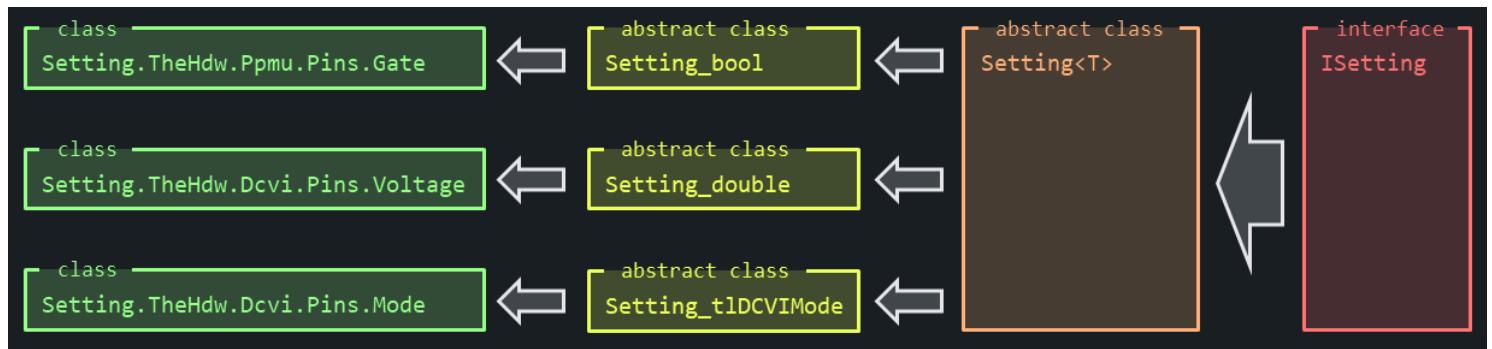
Setups can be defined in code or may be read in from separate setup files (JSON) once before validation.

Versatile Settings

A **setup** is a collection of **settings**, which follow a common structure while offering the most versatile functionality:

- **Feature** - the instrument / DIB / device feature it controls.
- **Pins** - the pin names this setting applies to (if any).
- **Value** - the setting value. Strictly and specifically typed (like **double** for a force voltage setting or **t1UtilBitState** for an utility bit state).
- **Set delegate** - the code required to apply this setting to the instrument / DIB / device
- **Read delegate** - the code required to retrieve the actual setting state from the instrument / DIB / device. Optional - may not always be possible.

Different values for the same feature on different pins are realized by the creation of dedicated settings. These are implemented by following a strong interface implementation / inheritance model. Common functionality is provided centrally & generically and shared with the specific implementation. This avoids copy / paste or consistency issues, admittedly at the slight cost of a non-trivial implementation:



Fully Specified, Incrementally Applied

The setups retain a cached state per feature, so that they can determine if a setting needs to be re-applied or not. Incremental & partial setups are supported, and the logic is smart enough to only apply the minimum needed to the system. Settings that are defined for multiple pins will on-the-fly determine the subset of pins that require a change, pins already programmed to the target value are omitted and the remaining ones are programmed in a single statement. If all pins are at the target value already, no programming statement is issued.

Audit Mode

Determining the deltas to be programmed is made efficiently by comparing the target setting with cached values. No expensive hardware reads are performed for that. If however, settings are modified by bypassing the cache, the mechanism would break.

To detect that, an **Audit Mode** is available. Once enabled, actual hardware reads are performed to confirm caches' correctness as the test program executes. Identified violations are reported.

i NOTE

For certain instrument settings, comparing driver readback against the cache isn't trivial. The reason may be cal factors being applied, quantization effects or instrument ranges automatically picking the next available hardware range on arbitrary inputs. Using tolerances and instrument / feature specific logic for the comparison would be unreliable and error prone.

The **DoubleReadCompare** feature adds a different compare strategy to these settings. In AuditMode, the driver value is read back twice, first as-is and then again after re-applying the target value. In case both reads match, the expected value was obviously already present in the hardware. Otherwise the cache was out of sync - regardless of how the value is altered on the way when being sent to that specific driver node.

The implementation of this feature is transparent to the user and automatically applied for those settings in SetupService that require it.

The **Audio Mode** is intended to be used prior to releases or correlation runs, since the hardware readback adds significant overhead. **Audit Mode** runs could be required by process or enforced in dev-ops pipelines.

Resolve a Cache Conflict

Three ways exist to resolve a cache conflict:

- Manually Revert the changed setting at the end of the test method. This option is preferred if the setting is rather unique in the program.
- Change the code to apply the setting via an additional **Setup** instead of manually. Setups can be small and incremental, and multiple can be applied within a test method.
- Notify the **SetupService** about the change by calling the static method `.SetCache()` on any setting. Only the cache is being updated without interacting with hardware.

IMPORTANT

In case of cache conflicts, it's critical to apply the resolution within the offending test. Otherwise a dependency between tests would be created, and changing the sequence or adding other tests in-between could break the model.

Confirm the System is in the Expected State (Future Enhancement)

Some users have asked for a way to confirm a specific setup at the moment a measurement is being made or a pattern is started. This is to make sure that no critical condition is altered after incremental edits to released test programs and rounds of test time optimizations. This check would only compare the hardware state with the expectation and report any deviations, and also be run per request as part of a release process.

Such a feature ("TCM - Test Condition Monitor") had been requested by multiple customers in the past independently from a **SetupService**, but the infrastructure could support both use cases easily.

Validation (Future Enhancement)

The Validation feature of the SetupService ensures the integrity and correctness of user-defined setups. During the validation process, the service performs the following checks:

- Pin Existence: Verifies that all pins specified in the setups are defined on the pinmap sheet.
- Instrument Type Consistency: Ensures that within each setting of a setup, all referenced pins belong to the same instrument type.

- Value Range Verification: Confirms that all specified values, where possible, are within the acceptable ranges of the setting.
- Duplicate Setup Detection: Identifies and flags duplicate setups to prevent redundancy.

Use Model

Assume the case where two different instrument parameters need to be controlled via setups: utility bits and DCVI force voltages. Defining a first setup in code could be done by:

Defining Setups

```
Setup normal = new("Normal");
normal.Add(new Csra.Settings.TheHdw.Utility.Pins.State(tlUtilBitState.Off, ["K2", "K3",
"K5", "K6"]));
normal.Add(new Csra.Settings.TheHdw.Utility.Pins.State(tlUtilBitState.On, ["K1",
"K4", "K7"]));
normal.Add(new Csra.Settings.TheHdw.Dcvi.Pins.Voltage(0.990, ["dcvi1"]));
normal.Add(new Csra.Settings.TheHdw.Dcvi.Pins.Voltage(2.850, ["dcvi2", "dcvi3"]));
Services.Setup.Add(normal);
```

In this example, a `Setup` called `Normal` is created. Certain utility bits are specified to be `Off`, while others need to be `On`. Two different voltage levels are defined for the three DCVI pins `dcvi1`, `dcvi2` and `dcvi3`.

In the same way, a second setup called `TestMode` is created with different settings.

```
Setup testMode = new("TestMode");
testMode.Add(new Csra.Settings.TheHdw.Utility.Pins.State(tlUtilBitState.Off, ["K1",
"K2", "K3"]));
testMode.Add(new Csra.Settings.TheHdw.Utility.Pins.State(tlUtilBitState.On, ["K4", "K5",
"K6", "K7"]));
testMode.Add(new Csra.Settings.TheHdw.Dcvi.Pins.Voltage(2.2, ["dcvi2"]));
Services.Setup.Add(testMode);
```

NOTE

Setups don't have to be "complete" or "consistent" with others. Like the `TestMode` setup only cares about the voltage on `dcvi2` but ignores both `dcvi1` and `dcvi3`.

Adding a specific setting to a setup means that the given values only relate to the pins listed. Other pins or other features don't matter here and are neither touched nor concerned.

Applying Setups

Within a test method, setups can easily be applied:

```
[TestMethod]  
public void Services.Setup1() { // Test #1  
    Services.Setup.Apply("Normal");  
    // do some DUT testing  
}
```

In verbose mode, the `Services.Setup.Apply()` is chatty, reporting to the output window what it did:

Test SetupService1

```
Apply Setup: Normal  
State @ 'K2, K3, K5, K6' -> tlUtilBitState.Off: no action needed  
State @ 'K1, K4, K7' -> tlUtilBitState.On: need to program 'K1, K4, K7'  
Voltage @ 'dcvi1' -> 0.990V: need to program 'dcvi1'  
Voltage @ 'dcvi2, dcv3' -> 2.850V: need to program 'dcvi2, dcv3'
```

All defined **settings** of the **setup Normal** are applied in the specified sequence. For the first setting (turn utility bits off), no actions are needed as they are off already.

i NOTE

After a post-job-reset, most instrument parameters are reset to a known default state. For settings resembling tester instrument features, the **SetupService** is aware of these defaults and automatically updates (resets) its caches at the beginning of a test program run.

Now let's assume the next test in our example requires the **setup TestMode** to be applied:

```
[TestMethod]  
public void SetupService2() { // Test #2  
    Services.Setup.Apply("TestMode");  
    // do some DUT testing  
}
```

Only the differing settings are being sent to the hardware:

Test SetupService2

```
Apply Setup: TestMode  
State @ 'K1, K2, K3' -> tlUtilBitState.Off: need to program 'K1'  
State @ 'K4, K5, K6, K7' -> tlUtilBitState.On: need to program 'K5, K6'  
Wait -> 0.001s:  
Voltage @ 'dcvi2' -> 2.200V: need to program 'dcvi2'
```

If the third test now calls for the same **setup TestMode**:

```
[TestMethod]
public void SetupService3() { // Test #3 (reusing same setup)
    Services.Setup.Apply("TestMode");
    // do some DUT testing
}
```

no more action is needed - all settings are already where they need to be and no hardware programming is done:

Test: SetupService3

```
Apply Setup: TestMode
State @ 'K1, K2, K3' --> tlUtilBitState.Off: no action needed
State @ 'K4, K5, K6, K7' --> tlUtilBitState.On: no action needed
Wait --> 0.001s:
Voltage @ 'dcvi2' --> 2.200V: no action needed
```

The fourth case performs some direct hardware modifications before re-applying **setup TestMode** again:

```
[TestMethod]
public void SetupService4() { // Test #4 (same setup, but some hardware changed)
    TheHdw.Utility.Pins("K1").State = tlUtilBitState.On;
    TheHdw.DCVI.Pins("dcvi2").Voltage.Value = 1.9;
    Services.Setup.Apply("TestMode");
    // do some DUT testing
}
```

Now the cache is confused as it didn't get notified of the hardware state changes made behind its back. It still believes no action is needed:

Test: SetupService4

```
Apply Setup: TestMode
State @ 'K1, K2, K3' --> tlUtilBitState.Off: no action needed
State @ 'K4, K5, K6, K7' --> tlUtilBitState.On: no action needed
Wait --> 0.001s:
Voltage @ 'dcvi2' --> 2.200V: no action needed
```

However, when using **Audit Mode** to re-apply **setup TestMode**:

```
[TestMethod]
public void SetupService5() { // Test #5 (audit mode finds such changed)
    Services.Setup.AuditMode = true;
    Services.Setup.Apply("TestMode");
```

```
// do some DUT testing  
}
```

the discrepancies are detected and flagged:

Test SetupService5

Apply Setup: TestMode

```
Hardware mismatch: pin K1 @ site 0 is tlUtilBitState.On but expected to be tlUtilBitState.Off  
Hardware mismatch: pin K1 @ site 1 is tlUtilBitState.On but expected to be tlUtilBitState.Off  
State @ 'K1, K2, K3' --> tlUtilBitState.Off: need to program 'K1'  
State @ 'K4, K5, K6, K7' --> tlUtilBitState.On: no action needed  
Wait -> 0.001s:  
Hardware mismatch: pin dcvI2 @ site 0 is 1.900V but expected to be 2.200V  
Hardware mismatch: pin dcvI2 @ site 1 is 1.900V but expected to be 2.200V  
Voltage @ 'dcvI2' --> 2.200V: need to program 'dcvI2'
```

Applying Multiple or Combining Setups

Users may decide to group related settings in separate setups, and then apply multiple to get the tester / device / interface hardware into the desired state. This may help with maintainability and reduce the total number required when combinations are needed.

The method `Services.Setup.Apply()` accepts a list of comma separated values, which are applied in the specified sequence. Redundant or conflicting settings (`Gate` off and then on again in a subsequent setup) are executed as specified, because it could be the intention of the author to preserve that order, for instance to apply a device reset.

Side Effects and Other Aspects

In IG-XL, persistently stored data causes cross-process communication overhead when running in blue button debug mode: the data has to be serialized to keep the DRH and the Excel process in sync. IG-XL automatically takes care of that, but a design focusing on minimum required data volume and simple structures helps minimize the impact.

A looming risk of features that bury complex functionality under the hood and expose an "always and automatically do the right thing" use model is debuggability. Designed for the "Happy Day Scenario", it may get tricky for users to trouble-shoot and figure out what's going on if things don't play out as intended. The verbose mode should provide valuable insight in such cases, but additional debug features may need to be considered as users run into challenges or identify additional requirements.

Certain events happen throughout the life-time of an IG-XL job, which the **SetupService** must reliably tie in to: settings need to be initialized once and will need retain its information with the life-time of the job. A post-job-reset will affect instrument settings, which the **SetupService** cache has to get notified off. A robust design here is essential for reliable functionality under all conditions.

Finally, more formal report outputs may be desired for the audit mode feature. Customer engagement will help identify requirements, but the design is open and extensible for additional functionality.

Supported Features

The selection of features is made based on priority and relevance for typical test programs. A brute-force code reflection based query of all existing nodes in the IG-XL driver language is difficult due to complexity (~800 nodes for DCVI alone), redundant and convoluted language implementation.

Detailed information on the supported features per instrument and their implementation strategy is shown on the following pages. The structure closely follows the IG-XL driver language, with a separate table per base node (the SetupService **is** platform specific)

- [DCVI instrument nodes](#)
- [DCVS instrument nodes](#)
- [Digital instrument nodes](#)
- [PPMU instrument nodes](#)
- [Utility instrument nodes](#)

The required types for the settings use there are listed here:

- [Setting Types](#)

 **NOTE**

The per instrument catalog does not reflect a final state - functionality is added on a by-need basis. This is a work-in-progress area.

Implementation

A conceptual description of the implemented code will follow ([https://github.com/TER-SEMITEST-
InnerSource/cs-reference-architecture/issues/689](https://github.com/TER-SEMITEST-InnerSource/cs-reference-architecture/issues/689)).

Unit Testing

The core functionality of the SetupService has far-going unit test coverage. However - designed for an open and extensible use model - unit test coverage for those extensions is required as well. For consistent coverage, the following strategy is defined:

- `Setting_Xyz : Setting<Xyz>`
 - `Serialize()`
 - enums: test all, invalid (cast)

- numeric types: test corner cases (-maxneg, small neg, 0, happy day pos, maxpos)
- **string**: normal, large, veryvery large, string.empty, null, (Unicode? No)
- **bool**: true/false
- **Compare()**
 - equal, smaller, larger
 - corner / special, tricky cases
- **Feature : Setting_Xyz**
 - Constructor
 - all args find their way to the object?
 - initstate set, unit, settling and initMode set ---> really?
 - **SetAction / ReadFunc** defined?
 - **_staticCache** connected?
 - **SetAction**
 - set corner cases from above and read back via test harness
 - **ReadFunc**
 - set happy day via SetAction IG-XL / readback ---> set another happy day / readback
 - **Set**
 - set happy day / readback --> set another happy day / readback
 - check cache

Alternatives Considered

n/a

Open Questions

- likely should integrate with the **Pins** objects. But that'll cause a dependency. Is that what we want?
Implications?
 - <https://github.com/TER-SEMITEST-InnerSource/cs-reference-architecture/issues/189>
- add validation support. Bounds checking might be possible. Should we propose defining the **setups** at validation? Prior to that?
 - <https://github.com/TER-SEMITEST-InnerSource/cs-reference-architecture/issues/190>
- add site capability (different values per site for scenarios where that is needed)
 - <https://github.com/TER-SEMITEST-InnerSource/cs-reference-architecture/issues/192>

Single- and MultiCondition

Data broadcast to tester channels is a fundamental principle in IG-XL: whether the same voltage setting needs to be sent to multiple pins, the drivers will take care of that and handle the broadcasting transparent to the user in the background. True is also: broadcast isn't an option whenever different settings are needed.

On the user interface, such different settings often result in exuberant code to determine and manage these conditions in test code plus `For` loops to apply them. Trying to be smart and optimizing the number of calls required adds even more complexity, resulting in error prone code that is hard to test and maintain.

C# Reference Architecture uses a common concept for handling single- and multi condition data across pins and pin groups, consistently implemented in areas that need this functionality. A set test blocks and other features allows for an abstraction level similar to the single-condition case, removing friction and clutter from the user.

PinLists and PinGroups

IG-XL supports the concept of Pins, PinGroups and PinLists. For this feature it is important to understand the differences:

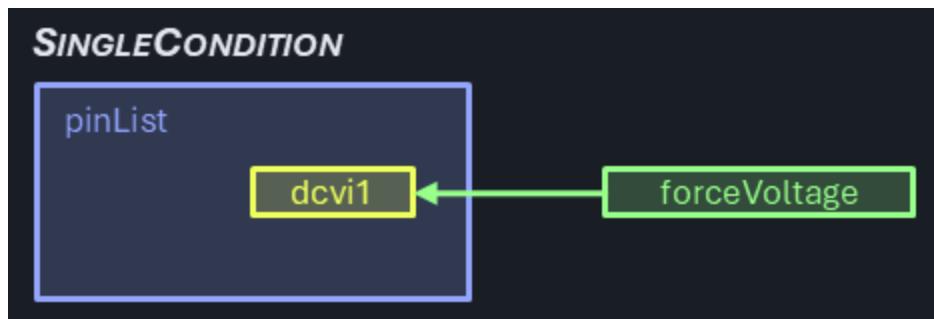
- Pins are the atomic units found on the PinMap and ChanMap sheet. They typically resolve to a single tester channel for every DUT site.
- PinGroups are collections of Pins (and other pin groups, nesting is supported), also defined on the PinMap sheet. Only pins of a common type can be part of a PinGroup, which is verified during validation.
- PinLists are loose collections of Pins, PinGroups and other PinLists and typically handled in comma separated strings (CSV). Arbitrary nesting and types are supported. PinLists can either be expressed as a `string` or by using the `PinList` type (which is mostly an alias of `string`). Pin based instrument drivers typically accept pin lists as input and will resolve to hardware resources internally. PinLists are not formally defined in IG-XL or validated by default, they are rather used as literals on sheets (like `TestInstance`) or hardcoded in code.

IMPORTANT

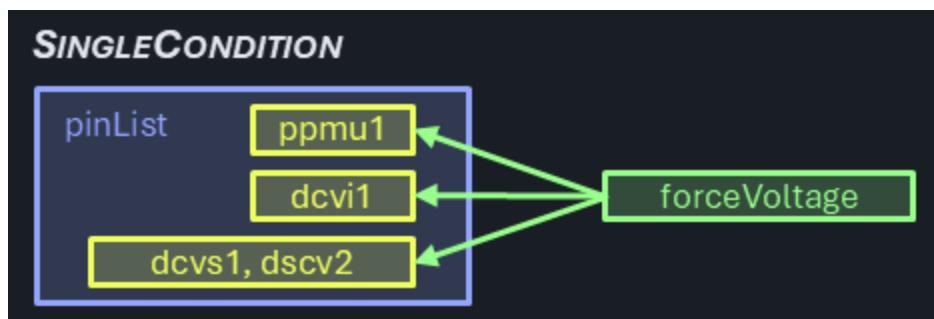
This feature exclusively applies first level splitting of comma separated value lists (CSV). Each of the resulting elements is considered a separate entity (be it Pin or PinGroup) for the multi condition case. No further resolution of PinGroups is attempted. Condition counts not matching the split element count result in an error.

Functionality

For reference, the single condition scenarios present when a single parameter is sent to a single pin:



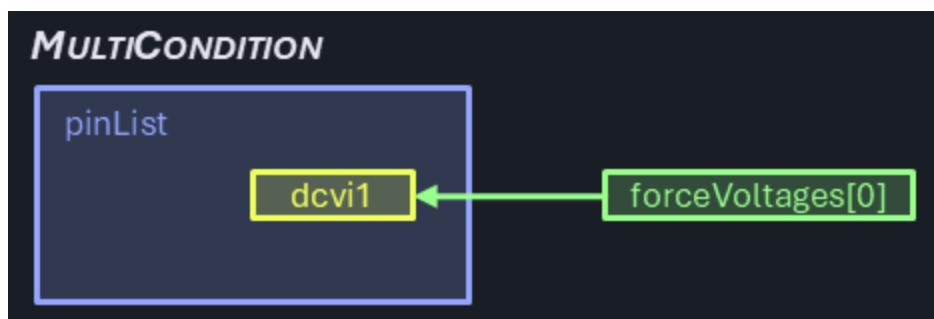
Even when multiple pins are targeted, it's still a single condition case:



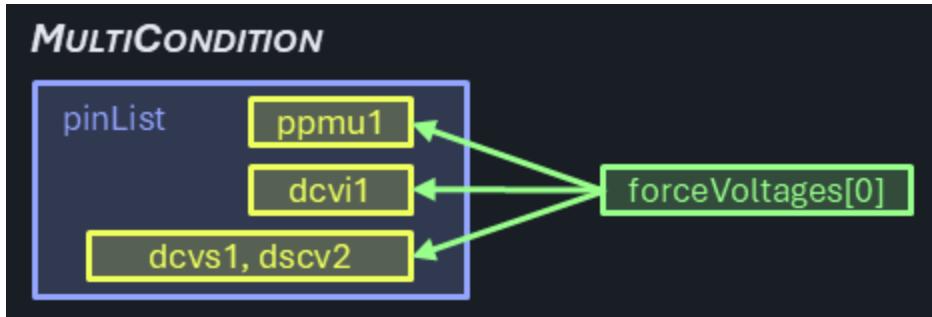
Specific conditions for different pins is the typical scenario for the multi-condition case:



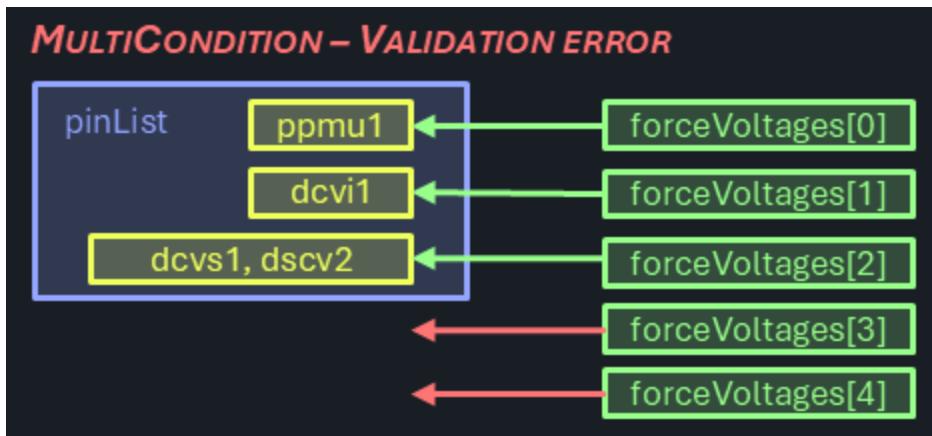
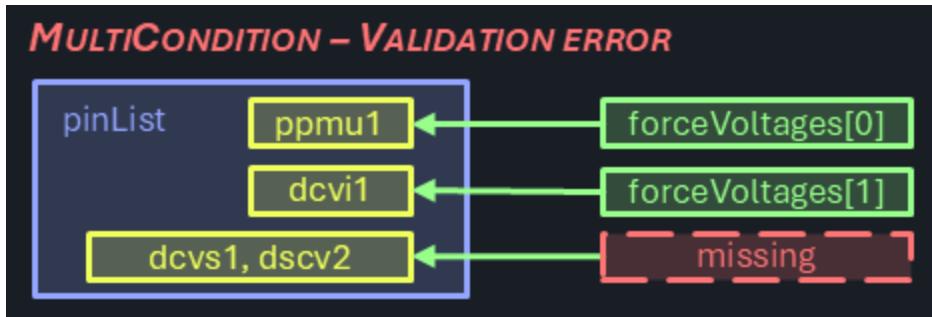
All cases when the number of pins matches the number of conditions must be correctly handled by the multi-condition scenario. That includes the 1 pins / 1 condition case, even if that's close to the single pin / single condition scenario. Note that in this case the parameter is provided as a collection with a single element vs. a scalar value:



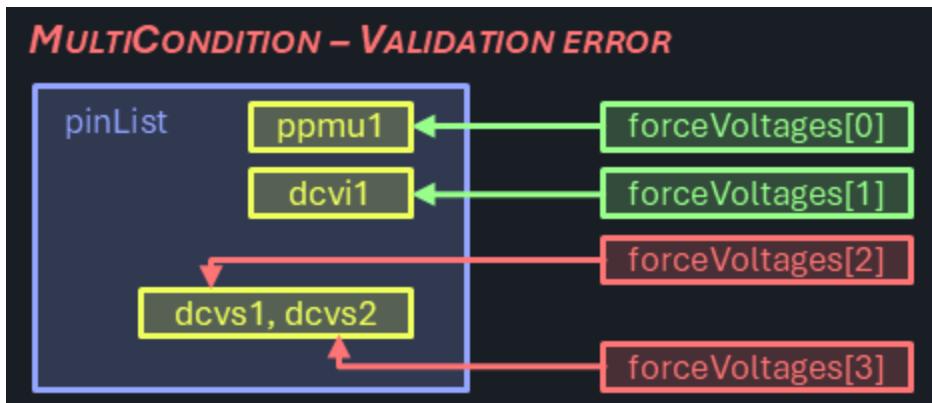
The multi-condition scenario still supports uniform parameters - if one is provided, it's shared across all pins:



Multi-condition cases, where the number of parameters provided does not match the pin group count and isn't uniform (1) are unsupported:



Even the case where the number of parameters would theoretically match the flattened pin count is not allowed. Use flat pin lists in that case instead:



Single Condition Test Method

Let's assume this (fictitious and simplified) test method:

```
[TestClass(Creation.TestInstance)]
public class SingleConditionTesting : TestCodeBase {

    Pins _pins;

    [TestMethod, CustomValidation]
    public void ForceV(PinList pinList, double forceVoltage, double voltageRange, double
currentRange) {

        if (TheExec.Flow.IsValidating) {
            _pins = new(pinList);
        }

        if (ShouldRunBody) {
            TheLib.Setup.Dc.ForceV(_pins, forceVoltage, voltageRange,
currentRange);
        }
    }
}
```

During validation, the `Pins` object is created and persistently stored. It is used in the `Body` of the test method in the sole test block to force a voltage.

Multi Condition Test Method

The same concept applied to a multi-condition case results in the following test method:

```
[TestClass(Creation.TestInstance)]
public class MultiConditionTesting : TestCodeBase {

    Pins[] _pinGroups;
    double[] _forceVoltages;
    double[] _voltageRanges;
    double[] _currentRanges;

    [TestMethod, CustomValidation]
    public void ForceV(PinList pinList, string forceVoltages, string voltageRanges, string
currentRanges) {

        if (TheExec.Flow.IsValidating) {
            _pinGroups = TheLib.Validate.SplitMultiCondition(pinList, p => new Pins(p));
        }
    }
}
```

```

        _forceVoltages = TheLib.Validate.SplitMultiCondition(forceVoltages,
double.Parse, _pinGroups.Length);
        _voltageRanges = TheLib.Validate.SplitMultiCondition(voltageRanges,
double.Parse, _pinGroups.Length);
        _currentRanges = TheLib.Validate.SplitMultiCondition(currentRanges,
double.Parse, _pinGroups.Length);
    }

    if (ShouldRunBody) {
        TheLib.Setup.Dc.ForceV(_pinGroups, _forceVoltages,
_voltageRanges, _currentRanges);
    }
}
}

```

Because the options for instance --> test method argument types are limited (arrays or lists are not available), data collections must be passed as string. Comma separated value lists (CSV) are commonly used in IG-XL for this use case, like for pin lists.

These strings must be parsed, which is done during validation time in dedicated **Validation** blocks. Besides splitting and parsing, that can also check for a matching element count (uniform ==> 1, multi-condition ==> must match reference) and raise a validation error otherwise.

The strings are converted into specifically typed arrays, while a fitting parser can be specified to support flexible types. Those arrays are stored in the test method, so that the parsing isn't affecting test time.

Multi Condition Test Block

The test block **ForceV** called to perform the action in the **Body** section of the test method above uses the exact same use model as in the single condition case. It's offered as an overload:

```

public static void ForceV(Pins[] pinGroups, double[] forceVoltage, double[] voltageRange,
double[] currentRange) {
    if (forceVoltage.Length == 1 && voltageRange.Length == 1 && currentRange.Length ==
1) {
        TheLib.Setup.Dc.ForceV(Pins.Join(pinGroups), forceVoltage[0],
voltageRange[0], currentRange[0]);
    } else {
        for (int i = 0; i < pinGroups.Length; i++) {
            TheLib.Setup.Dc.ForceV(pinGroups[i], forceVoltage.SingleOrAt(i),
voltageRange.SingleOrAt(i), currentRange.SingleOrAt(i));
        }
    }
}

```

NOTE

Test Blocks are exclusively called through the official path `TheLib` which goes through the versioned interfaces instead of directly referencing the implementation, even in the case where both implementations are placed right next to each other. This is done to showcase recommended practice when users copy the code for custom implementations in their own projects, where interfaces are the only accessible option.

First, it determines if all provided parameters are uniform (array length == 1). In that case it can combine the pins from all pin groups and call the single condition test block only once.

Otherwise, the single condition test block is called per pin group with its associated parameters. To support a mix of uniform and specific parameters, the `.SingleOrAt()` extension method is used for a concise syntax, returning the index element (specific case) or the single element if only one exists (uniform case).

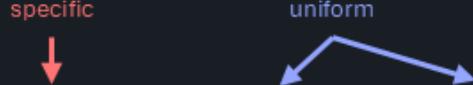
NOTE

It may be surprising why a single condition case is prominently offered and specially treated here - can't user simply use the single-condition flavor of the test block instead?

Real test methods will have a series of test blocks that are required to perform a specific test, and only some of the test method's parameters may need to have specific conditions per pin. And since test blocks may not require all of the test methods parameters for functionality, it can't be determined at design time which will for a truly flexible multi-condition test method.

The proposed concept buries that logic under the hood but offers optimized performance in either case. The increased design effort is covered by a standardized design pattern, that can be simply followed by test block authors.

```
// MultiCondition
public void TestMethod(PinList pinList, string values1, string values2, string values3) {
    TestBlockA(_pinGroups, _values1, _values2) // needs to run in loop because of _values1
    TestBlockB(_pinGroups, _values2, _values3) // can program all pins together
    TestBlockC(_pinGroups, _values1, _values2, _values3) // needs to run in loop because of _values1
}
```



Alternatives Considered

List<T> or T[] arrays

For the resolved value collections, either `List<T>` or `T[]` arrays objects could be used.

- `List<T>` are part of GenericCollections, a powerful set of types that are prominently supported. They offer dynamic extension or shrinking at the cost of a slight performance and memory footprint overhead.
- `T[]` arrays are fixed size collections with the best performance but no (direct) support for dynamic size changes. They could be considered inferior by strong VBA veterans as that was the only choice for collections there, and may have been associated with clumsy implementations.

Since both offer the advanced collection and LINQ use model in .NET, and the benefit of dynamic resizing is not very relevant in this use case, arrays are selected.

This decision shall not be interpreted as a general preference for collection objects in the C#RA project. .NET offers a variety of collection types as part of the `System.Collection.Generic` namespace, but these are not meant as a replacement for regular arrays. The decision for the best type should be made on a case-by-case basis, the following guidelines might help:

- fast access & minimum footprint? → `T[]` arrays
- dynamic size? → `List<T>`
- fast insert/delete? → `LinkedList<T>`
- unique elements? → `HashSet<T>, SortedSet<T>`
- fast lookup? → `Dictionary<K, V>, HashSet<T>`
- FIFO? → `Queue<T>`
- LIFO? → `Stack<T>`

Array size / general test block parameter validation

Argument bounds checking and validation is a general recommendation for good and robust software. However, the additional code executed can add significant overhead to runtime. IG-XL uses the concept of validation to perform checks on information that is static for the test program runs upfront. This is true for sheet data, config limitations and can be extended to test code by using the `[CustomValidation]` attribute. It is however an opt-in model, with a due diligence requirement for test code authors to perform checks on critical inputs. Those are not enforced or guaranteed by the system.

In favor of performant test execution, C# Reference Architecture uses extensive validation time checking in test modes and avoids additional argument checking in test blocks. Specifically, in this case, multi-condition array arguments are checked for length `==` pin group count (specific) or `== 1` (uniform) at validation time. Inside the test blocks only a check for `== 1` is required to discern the two.

Transaction Service

Manually implementing device communication using hardcoded patterns or custom-designed digital source and capture signals is both labor-intensive and increasingly outdated, particularly for high-complexity ("big-D") devices. Modern design workflows that incorporate device testing enable this complexity to be abstracted into standardized data communication protocols, with low-level signal manipulation (bit-bashing) handled transparently by the underlying system.

NOTE

The setup part of Transaction Service communication is deferred to a later point. In the first tier, the APIs for the transactions themselves are offered. A generic setup use model is added at a later point, until then users are deferred to instrument specific language for this task.

PortBridge

On IG-XL based platforms, PortBridge is the product providing that abstraction use model for test programs. It's highly flexible and configurable on the user level, and interacts with hardware features like ProtocolAware or PatternModify in the most efficient way.

PortBridge is an optional product, which users may or may not choose to use. Licensing cost is only one aspect; in some cases, customers rely on custom infrastructure that has matured over years and is tightly integrated into their design processes to work with multiple competitive tester platforms.

IMPORTANT

C#RA needs to support setups with or without PortBridge, at least from a use model perspective. Functionality and capability may differ, meaning that C#RA does not have to back-fill all PortBridge features in setups choosing to go without. Basic functionality should be covered though.

In case of PortBridge being available in a configuration, users may still want to decide to implement with or without on a by-test basis.

Use Model

Register Map

Typically, the device's address space is marked up with speaking names for registers (entire memory locations at the device's native bit width, i.e. 32bit) and fields (usually a sub-set of a single register, like 8 bits used for a 256 step voltage reference trim). That information is defined in the chip design tools and

commonly available in an interchangeable data format like XML. The test program reads this register map definition in order to allow interacting with the device via meaningful descriptors.

Register Maps may be used to apply device hardware abstraction. For example could the string `VREF_TRIM` generally refer to the voltage reference trim code field even if located at different physical memory locations in different devices. By using the abstract name to access that data, the same code / transaction language may be used for any device following that scheme.

Register Maps typically contain the following information:

- register: name and address
- field: name, register, start bit (LSB), bit width (sometimes bit order / masking and multi-register-span info)
- default / reset values (sometimes these may differ)

The TransactionService is designed in a generic way to open up support for non-`int` situations, which will become prevalent once IG-XL enables SiteGenerics support for those, too. This is currently planned for IG-XL 11.10.

Shadow Register

Shadow registers are register data structures mirrored to the tester computer memory reflecting the device's memory structure. It can be considered a cache.

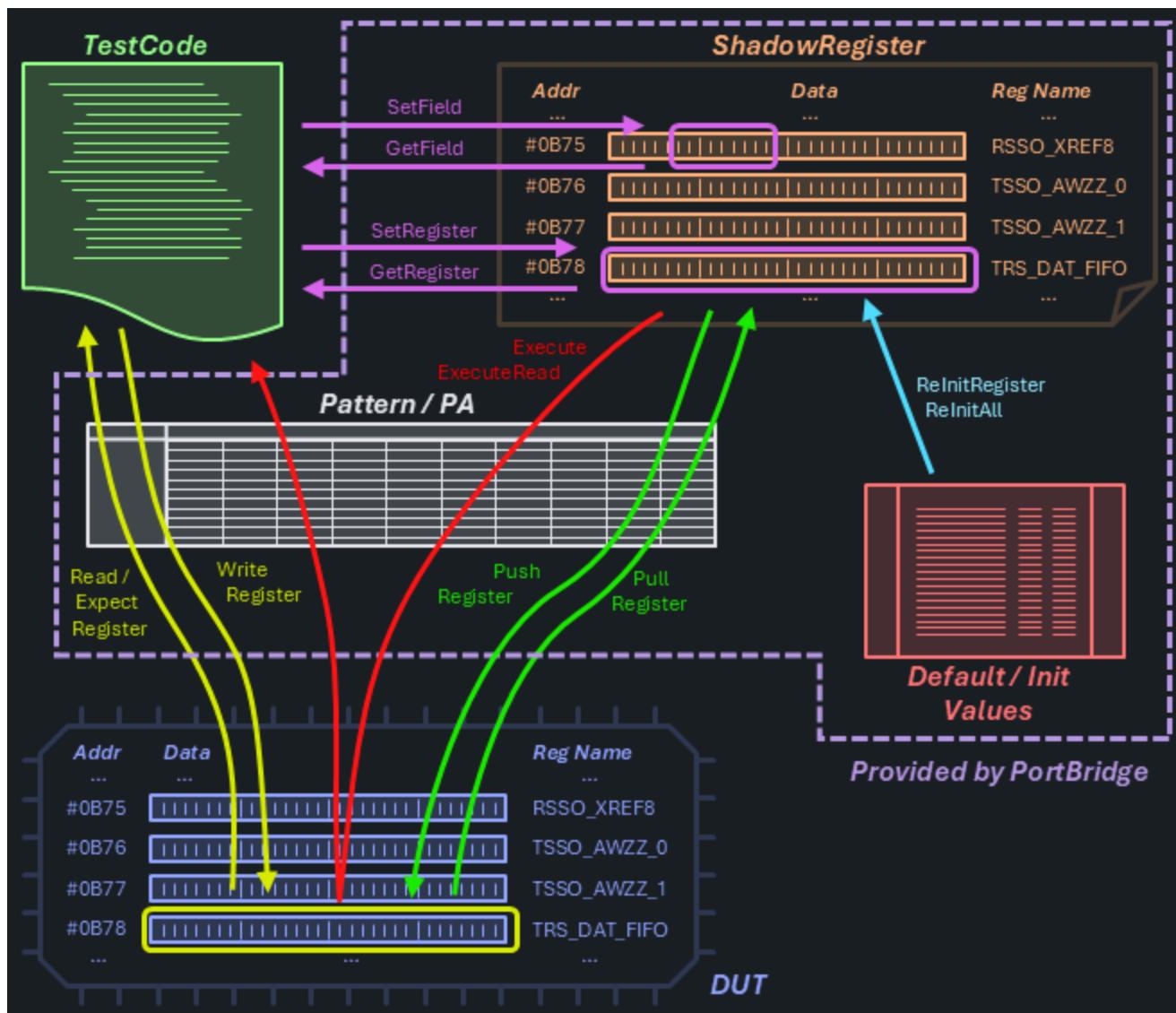
The shadow register provides site-aware capability. This can be achieved by maintaining a full copy per site. Alternatively, because the vast majority of information will typically not need this capability, the shadow register may be considered site-agnostic and only the site-aware activity is deferred into a dedicated, separate structure.

Multiple Ports

Some devices may have multiple register maps, be it to facilitate separate physical memory blocks or different access ports or protocols. C#RA supports this case, with an (optional) port argument in the transaction calls. If left empty, the (configurable) default port is used. This helps with a simplified use model for the common scenario of a single port device.

Data Paths

The following paths exist for data at runtime of a test program:



To easily distinguish the entities, clear naming notations are used. All operations work transparently for the selected sites.

TestCode <--> Shadow Register: Get / Set Notation

I/O operations with the Shadow Register can be executed at any time without physical device interaction. These operations are often executed in bulk for increased efficiency.

GetField

- `Site<T> GetField<T>(string register, string field, string port = "");` - read (bit masked & shifted) data from a shadow register field

SetField

- `void SetField<T>(string register, string field, T data, string port = "");` - write site-uniform (bit masked & shifted) data into a shadow register field
- `void SetFieldPerSite<T>(string register, string field, Site<T> data, string port = "");` - write site-specific (bit masked & shifted) data into a shadow register field

GetRegister

- `Site<T> GetRegister<T>(string register, string port = "");` - read data from a shadow register

SetRegister

- `void SetRegister<T>(string register, T data, string port = "");` - write site-uniform data into a shadow register
- `void SetRegisterPerSite<T>(string register, Site<T> data, string port = "");` - write site-specific data into a shadow register

Shadow Register <--> DUT: Pull / Push / Execute Notation

Actual device communication is handled via a digital / pattern subsystem allowing I/O operations on a register basis.

PullRegister

- `void PullRegister(string register, string port = "");` - transfer data from the device into the shadow register

PushRegister

- `void PushRegister(string register, string port = "");` - transfer data from the shadow register to the device

TestCode <--> DUT: Read / Expect / Write Notation

Direct DUT transactions are often implemented via digital transaction instruction language and executed as part of patterns / modules. These commands are offered to complete the use model, they may be helpful in debugging or troubleshooting scenarios.

ReadRegister

- `Site<T> ReadRegister<T>(string register, string port = "");` - read data from a device register

ExpectRegister

- `Site<bool> ExpectRegister<T>(string register, T data, string port = "");` - read data from a device register and compare with a site-uniform expectation
- `Site<bool> ExpectRegisterPerSite<T>(string register, Site<T> data, string port = "");` - read data from a device register and compare with a site-specific expectation

WriteRegister

- `void WriteRegister<T>(string register, T data, string port = "");` - write site-uniform data into a device register
- `void WriteRegisterPerSite<T>(string register, Site<T> data, string port = "");` - write site-specific data into a device register

Execute Transactions

Complete transaction modules including multiple write and read operations can be executed with a single command. Note that the write direction sends shadow register data to the device, where the read direction retrieves device data directly into test code.

Execute

- `void Execute(string module, string port = "");` - execute a transaction file (without reads)

ExecuteRead

- `List<Site<T>> ExecuteRead<T>(string module, int readCount, string port = "");` - execute a transaction file including reads

Re-Initialize the ShadowRegister and Others

Commonly at Start-of-Test, or when specific use cases exist, the shadow register content needs to be re-initialized.

ReInitRegister / ReInitAll

- `void ReInitRegister(string register, string port = "");` - set a shadow register to its default / init value
- `void ReInitPort(string port);` - set all shadow registers of a port to their default / init values
- `void ReInitAll();` - set all shadow registers of all ports to their default / init values
- `string DefaultPort {get; set;}` - read/write the default port to be used when omitted in the methods above

Implementation

The use of PortBridge will add an external dependency to C#RA - and the fact that this needs to be conditional makes things not easier.

The model for [conditional dependencies](#) in C#RA is followed with these specifics:

Entity	Term for PortBridge
custom MSBuild property	<code>PortBridgeEnabled</code>
indicator file	<code>PortBridgeEnabled.txt</code> located in VS solution folder
compiler constant	<code>PORTRIDGE_ENABLED</code>

IMPORTANT

By default, the repo does not have the `PortBridgeEnabled.txt` file in the `src` folder, thus not enabling this feature. The use model follows an opt-in approach, where users actively select additional, optional features they want to use.

`PortBridgeEnabled.txt` has been added to `.gitignore` so that collaborators can permanently change their local coding environment without the risk to propagate this into the repository.

Alternatives Considered

The following options were considered but could not prevail due to the reasons stated.

Port Object Retrieved from Dictionary (= very simplified TransactionService)

```
// setup
Services.Transaction.SetupPort("portA", settingA, ... , ... , ...);

// single call
Services.Transaction.Port["portA"].SetRegister<T>(string register, T data)

// multiple calls
var port = Services.Transaction.Port["portA"];
port.SetRegister<T>(string register, T data);
```

Ups	Downsides
close to IG-XL syntax	need to introduce <code>TransactionService</code> and <code>TransactionPort</code> type
	types can't be versioned
	and unit testing would be more complicated
	no easy way to handle default ports

Fully Object Oriented

```
// setup
var port = new TransactionPort(setting, ,... , ... , ...);
Services.Storage.Add(port, "portA");
```

```
// regular call
TransactionPort port = Services.Storage.TryGet("portA");
port.SetRegister();
```

Ups	Downsides
very much the ".NET Way"	strong dependency from <code>StorageService</code>
	(public) object could have been messed with or replaced with something else
	types can't be versioned
	no easy way to handle default ports

Encode Port in Register Identifier

```
// setup
Services.Transaction.SetupPort("portA", settingA, ... , ... , ...);

// regular call - require "portA@reg123" register definition, with pre-@ part optional
Services.Transaction.SetRegister<T>(string register, T data)
```

Ups	Downsides
eliminate port information from transaction method signatures	potentially expensive string processing in every call
easy to handle default	port and register information doesn't typically exist in this format

Questions

- How to read / define the register map?
- Required to support register spanning fields and crazy encodings (inverted bits, strange order)?
- What PB version do we support? Multiple?
- control file location - is the `src` folder the best place?

Characterization Support

- XXX

Copilot Friendliness

- xxx

Brainstorming

- What could we do to make this easily digestible for our Copilot / AI?
 - Comments, comments, comments. Tell the AI all of the unstated assumptions and goals you have. Mixing English and code in the training data helps the AI learn the relationship.
- We're going to have documentation, explaining the context of the code. That'll be as XML comments right at the public API nodes, as well as prose documentation (markdown), which will reference in code snippet examples. Can your engine intake such mixed-media inputs?
 - yes
 - So the following is outdated content, but the format is very much what we'll have:
<https://crispy-adventure-kqje5zq.pages.github.io/architecture.html>. Can this be fed in? Like directly, without converting to anything?
 - Can this be fed in? Robust libraries exist for ingesting all sorts of standard formats like HTML, Markdown, JSON, XML, PDF, Word, code, etc. Format is no problem. The thing to keep in mind is that a code-generating model will be given English prompts describing what the user wants and will be expected to output code that implements the user's intent. To make this happen, we need training data that helps the model learn this English-code association. So in the GitHub document you linked, it would be better if the text that describes the code examples is clearly grouped together with the code and separated from everything else. This grouping can be done anyway way that's convenient, such as the HTML h2 header "Singleton Classes for Blocks" containing the text and matching code. If this HTML section contains other text that doesn't describe the code example, this makes it harder for us to process it into English-code training examples. It's this pairing that matters, not the exact mechanism by which the code and its description are paired. Make sense?
- Are we working on Copilot functionality that I can ask to create common programming tasks? Like ChatGPT, but who is aware of the IG-XL specifics, and can create meaningful test & dsp code?
 - No. We were told by Apps that automating tedious processes like characterization is more important than generating code. Also, we lacked training data. I expect this reference architecture will be a big enabler so we can finally progress on code generation.
- Are we considering VS integrated functionality that can help me code right in the IDE? Like GH Copilot, but with awareness of our IG-XL knowledge and not only limited to the code it sees in my VS project?
 - Yes, this is a prime candidate for C# test code generation use mode.

- We may have some degrees of freedom to make those things potentially easier. Or if it is only to avoid making them extra-hard. And we're just about to get started, so now would be the right time to consider ...
 - In a lot of cases, if the text description is short, comments in the code might be the easiest and most robust way to associate the code with a description.

Custom Validation

IG-XL offers convenient test instance validation via the `CustomValidation` attribute for test methods. During validation, every test instance using a test method decorated that way is called with its correct context. While no hardware can (and should!) be programmed in that phase, this mechanism can be used to:

- check input values for valid range (like voltages, currents)
- check pins / patterns / method delegates / setups are valid
- verify the requested combination of inputs is supported
- perform time consuming object initialization and input processing once only

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(PinList pinList, double voltage, double currentRange, double waitTime,
string setup = "") {

    if (TheExec.Flow.IsValidating) {
        // perform validation only tasks
    }

    ...
}
```

Object Initialization

Test parameterization is done in IG-XL by the flow controller handing the parameters into the test method at runtime. In IG-XL, that data is cached, and locally made available within the test method that way. While this is performance optimized for the case when data is used "as-is", some overhead may occur when it needs to be converted or processed before it can be used.

To avoid that, such processing can be done once only during validation and preserved for subsequent use. Because the input is static (= the same in every run), pre-processing results can be stored in class using test classes persistence:

```
[TestClass(Creation.TestInstance)]
public class Read : TestCodeBase {

    PatternInfo _patternInfo;
    Pins _pins;
    tlBitOrder _bitOrder;

    [TestMethod, Steppable, CustomValidation]
    public void Baseline(Pattern pattern, PinList readPins, int startIndex, int bitLength,
```

```

int wordLength, bool msbFirst, bool testFunctional,
    bool testValues, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        _pins = new Pins(readPins);
        _patternInfo = new(pattern, true);
        _bitOrder = msbFirst ? tlBitOrder.MsbFirst : tlBitOrder.LsbFirst;
    }

    ...
}

}

```

Parameter & Context Checking

In many cases constraints exist for parameters, like min/max levels, pin counts, pattern existence ... and well-designed, reusable TestMethods should offer helpful messages when these are not properly met. IG-XL supports test instance argument validation error reporting, and [that mechanism](#) can be used for custom test methods:

Test Procedure		Arg0	Arg1	Arg2	Arg3	Arg4
Type	Name					
.NET	ACME123_NET.TestClass1.TestMethod1	1	1			
.NET	ACME123_NET.TestClass1.TestMethod1	-1	+ -1			

IGXL: Argument 'posValue': value must be positive

The following blocks are offered for validation at `TheLib.Validate` as a starting point. This list may not be complete, and over time, additional validation features may (will!) surface, and the design goal is that these can easily be added following the common use model and utilizing the infrastructure created here.

Validating a Single Parameter

These blocks offer validation of specific parameters, and support test instance parameter cell highlighting in case of a fail:

Argument Specific Test Blocks	Functionality
<code>bool InRange<T>(T value, T from, T to, string argumentName)</code>	checks if a numeric value is between two bounds (including)

Argument Specific Test Blocks	Functionality
<code>bool GreaterOrEqual<T>(T value, T boundary, string argumentName)</code>	checks if a numeric value is greater or equal to a bound
<code>bool LessOrEqual<T>(T value, T boundary, string argumentName)</code>	checks if a numeric value is less or equal to a bound
<code>bool Pattern(Pattern pattern, string argumentName, out PatternInfo patternInfo)</code>	checks for valid pattern spec and creates the object (preferred over <code>new PatternInfo()</code>)
<code>bool Pins(PinList pinList, string argumentName, out Pins pins)</code>	checks for C#RA supported pin spec and creates the object (preferred over <code>new Pins()</code>)
<code>bool MethodHandle<T>(string fullyQualifiedNamespace, string argumentName, out MethodHandle<T> delegate) where T : Delegate</code>	checks for valid method handle spec and creates the object (used to be <code>GetDelegate</code> , preferred over <code>new MethodHandle<T>()</code>)
<code>bool MultiCondition<T>(string csv, Func<string, T> parser, string argumentName, out T[] conditions, int? referenceCount = null)</code>	checks multi-condition validity and creates the data array (used to be <code>SplitMultiCondition</code>)
<code>bool Enum<T>(string value, string argumentName, out T enumValue) where T : Enum</code>	checks if a string value can be parsed to the specified enum type and creates the enum value
<code>bool Setup(string setup, string argumentName)</code>	checks if a setup with that name exists

✖️ IMPORTANT

In order to flag the correct test method argument, IG-XL needs to be informed which one caused the fail via the `argumentName`. This is achieved through reflection performing a method argument look-up. For a robust solution, that doesn't break in case of a rename, use the `nameof()` operator.

By requiring an `argumentName` in these methods, users are nudged to use them for checks that are tied to dedicated arguments. The methods however will quietly tolerate empty strings and still correctly fail validation - and highlight that fact in the validation error message shown.

Validating Combinations of Parameters or Other Scenarios

Validation checks that can't be connected to a single parameter only, like an illegal combination of parameters, which would be fine by themselves, or a certain system context to be available can also be

validated. In that case a meaningful validation fail message needs to be provided to the user, clearly describing the **problem**, **reason** and **resolution** options:

Argument Agnostic Test Blocks	Functionality
<pre>bool IsTrue(bool condition, string problemReasonResolutionMessage, string argumentName)</pre>	checks for condition == <code>true</code> (fallback for ANY checks)
<pre>void Fail(string problemReasonResolutionMessage, string argumentName)</pre>	raises an unconditional validation error

All validation methods report potential issues directly to IG-XL, which collects them for a collided validation error report. To support dependent checks ("only if pins are valid, check if any of type XYZ are in there"), they return a boolean result for success.

NOTE

The language node `TheLib.Validate` was previously called `TheLib.Validation` - the rename aligns it with the test block action categorization described in the [test block language hierarchy](#).

Applied to the test method above, the validation section could look like:

```
[TestClass(Creation.TestInstance)]
public class Read : TestCodeBase {

    PatternInfo _patternInfo;
    Pins _pins;
    tlBitOrder _bitOrder;

    [TestMethod, Steppable, CustomValidation]
    public void Baseline(Pattern pattern, PinList readPins, int startIndex, int bitLength,
    int wordLength, bool msbFirst, bool testFunctional,
    bool testValues, string setup = "") {

        if (TheExec.Flow.IsValidating) {
            TheLib.Validate.Pins(readPins, out _pins, nameof(readPins));
            TheLib.Validate.Pattern(pattern, out _patternInfo, nameof(pattern));
            TheLib.Validate.GreaterOrEqual(startIndex, 1, nameof(startIndex));
            TheLib.Validate.GreaterOrEqual(bitLength, 1, nameof(bitLength));
            TheLib.Validate.InRange(wordLength, 1, 32, nameof(wordLength));
            TheLib.Validate.Setup(setup, nameof(setup));
            _bitOrder = msbFirst ? tlBitOrder.MsbFirst : tlBitOrder.LsbFirst;
        }
    }
}
```

```
 }  
 ...  
 }  
 }
```

Documentation

Good intentions for great documentation exist in every project, but often enough this task falls off the table. "*Functionality first and then documentation*" is a flawed process, and we want to do better here.

In fact, this team is starting off with documentation, by creating design docs like this before implementing. Maybe - along with a focus on unit testing - we're just inventing a new trend **Test and Documentation Driven Development (TDDD)** here!

Documentation = Source Code + ???

To avoid mismatches and outdated information, documentation must live very close to the source code and take context information into account. Where things change, documentation update requirements need to be obvious, easy to implement and ideally well supported with tools.

The same processes to collaborate, review and track issues is used as for the product code itself. The documentation generation process is fully autonomous, running as part of the product build & test pipeline. Checks automatically flag violations or discrepancies, guaranteeing complete, consistent and correct documentation for the user.

XML Comments

The [XML Documentation feature](#) is used for any publicly accessible interface:

- automatically enables [IntelliSense](#) features when using them
- checked for consistency (exists? required fields filled? arguments specified correctly? return type described?) at the build process
- single source of truth API documentation
- travels with the code, is always in sync
- written and maintained by the code author

Tags to Use

In C#RA, only those tags are used which source information into the IntelliSense service. That is a small subset of the ones existing. On the top level, these are:

- `<summary>` - one or two brief sentence to describe the function
- `<param>` - for every parameter: describe the parameter, include critical aspects ("zero based index", "non-empty string", ...)
- `<returns>` - for non-`void` methods: describe the return value
- `<typeparam>` - for generic methods: describe the meaning / functionality of the type

IMPORTANT

C#RA does not use the other tags supported for XML API documentation, like `<remarks>`, `<exception>`, `<example>`. Such further-going information is provided via [API doc extensions](#). The XML features to structure such rich documentation are limited, and would significantly inflate the source code.

Use the following guidelines to create consistent and efficient XML documentation:

-  **Do** use grammatically correct sentences in English language and end them with period (.). Parameter and return value descriptions may omit the verb if the use is obvious (/// `<param name="name">The pin name.</param>`).
-  **Do** use the `<cref>` tag when referring other types - doing this will provide context aware rename support ([Ctrl1-R-R](#)) and issue a compiler warning on mismatch.
-  **Do** use normal, English capitalization.
-  **Do** use indicative instead of imperative mood in summary and other places (/// `Creates a new object.` instead of /// `Create a new object.`)
-  **Do** add brief inline comments right at code features that might be unexpected to clarify it was a deliberate choice. Otherwise, code should strive for being self-explaining.
-  **Don't** use formatting or line breaks - they will be filtered out in Intellisense and make the text unreadable.
-  **Don't** use abbreviations or acronyms without need.
-  **Don't** repeat information about types or optional defaults. This information is automatically extracted from code and may get outdated if that changes.
-  **Don't** provide excessive information here, that should rather go into the prose documentation (Overwrite) pages.
-  **Don't** make excessive use of inline comments explaining the functionality. This information should go into the XML or override API comment section, so that it can be consumed for IntelliSense and the published documentation.

Here is a good example of how to **not** do it:

```
145 Reference |<inner class>| <hours> 12 authors | Changes
146
147     private static Type GetInstrumentType(string pin) {
148         ForEachSite<T>
149             if (pin.Lar)
150                 TheExec.Bet
151             switch (num)
152             {
153                 //case
154                 case 1:
155                     ForEachSite<Sub>(site As Integer) TheExec.Flow.TestLimit(site, ForceResults:=t1LimitForceResults.Flow), t1SiteType.Selected)
156                     Provide for each site operation function with exception handling
157
158                     VB.NET#1 Single Line Lambda Function Without Error Handler
159                     case 1:
160                         str
161                         int
162                         str
163                         ret
164
165                         VB.NET#2 Single Line Lambda Function With Lambda Error Handler
166
167                         ForEachSite<Sub>(site As Integer) TheExec.Flow.TestLimit(site, ForceResults:=t1LimitForceResults.Flow),
168                             Func<Exception, bool> errorHandlerFunc = null()
169                             void TestCodeBase.ForEachSite<Action<int>> perSiteAction, [TSiteType siteType := t1SiteType.Selected], [Func<Exception, bool> errorHandlerFunc = null()]
170
171                         ForEachSite<Sub>(site As Integer) TheExec.Flow.TestLimit(site, ForceResults:=t1LimitForceResults.Flow)
172
173                     VB.NET#3 Multiple Line Lambda Function Without Error Handler
174
175                     ForEachSite<Sub>(site As Integer)
176                         TheExec.Flow.TestLimit(site, ForceResults:=t1LimitForceResults.Flow)
177                         End Sub)
178
179                     VB.NET#4 Multiple Line Lambda Function With Error Handler
180
181                     private static
182                         return type
183                         ForEachSite<Sub>(site As Integer)
184                             TheExec.Flow.TestLimit(site, ForceResults:=t1LimitForceResults.Flow)
185                             Type_UF
186                             Type_UF
187                             Type_UL
188                             Type_UL
189                             Type_UL
190                             Type_UL
191                             Type_BU
192                             Type_BU
193                             Type_BU
194                             Type_BU
195                             Type_BU
196                             Type_BU
197                             Type_BU
198                         };
199                     VB.NET#5 Function Pointer Without Error Handler
200
201                     Public Sub SampleTestFunc()
202                         ForEachSite(AddressOf SampleSiteLoopFunc)
203                     End Sub
204
205                     Reference |<inner class>| <hours> 12 authors | Changes
206                     private class PinC<
207                         O References |<inner class>|
208                         public bool Equal()
209                             if (x == null)
210                             if (y == null)
211                             return x == y
212
213                         O References |<inner class>|
214                         public int Get()
215                             return obj.
216
217                     Public Sub SampleSiteLoopFunc(site As Integer)
218                         TheExec.Flow.TestLimit(site, ForceResults:=t1LimitForceResults.Flow)
219
220                     VB.NET#6 Function Pointer With Error Handler
221
222                     Public Sub SampleTestFunc()
223                         ForEachSite(AddressOf SampleSiteLoopFunc, , AddressOf ErrorHandlerFunction)
224                     End Sub
225
226                     Public Sub SampleSiteLoopFunc(site As Integer)
227                         TheExec.Flow.TestLimit(site, ForceResults:=t1LimitForceResults.Flow)
228                     End Sub
229
230                     Public Func<ErrorHandlerFunction>(Exception ex) As Boolean
231                         TheExec.ErrorLogMessage("Exception : " + ex.Message)
232                         Return AbortTest()
233                     End Func
```

DocFX

[DocFX](#) is used to publish to [GitHub Pages](#). It's the state-of-the-art solution for .NET projects, created by Microsoft for their Microsoft Learn library. It's open-source and easily deployable through GitHub Actions.

Feature Pages and More (Markdown)

Pages like this one is directly shown as a GitHub Page .

For a consistent style follow these guidelines:

- **✓ Do** target the "interested novice" for the right tradeoff between assuming too much prior knowledge and excessive explanations. Assume the reader has experience in the domain, but hasn't yet mastered the topic you're describing (why would they read that otherwise?).
 - **✓ Do** create a page outline before you fill in details. Use (hierarchical) headlines to structure your content.
 - **✓ Do** make use of pictures, animations, tables & mermaid diagrams. Shoot for the most compact way to convey your content ("one picture says more than 1000 words").
 - **✓ Do** consider all user types in the target audience, which includes humans and LLMs. While LLMs can understand markdown structured content like tables or mermaid diagrams well, comprehension of bitmap images is much less reliable, and should not be the only place critical information is conveyed. Describe such content in the **Alt-Text** field of image links, and aim for a crisp 1-3 sentence summary.

- **✓ Do** set a high standard for documentation. It is essential for users to successfully utilize the product.
- **✓ Do** compare how other, similar nodes are documented and follow their style.
- **✓ Do** use double-** (=bold) highlighting sparingly, avoid italics (* / _) and bold+italics (***/_____) for legibility and a consistent look and feel.
- **✓ Do** insert empty lines between different content types (headlines, flow text, bullet item groups, tables, ...) or paragraphs. The HTML renderer may not reliably work otherwise.
- **✗ Don't** use more than one first level # **Top Headline** in your document. It should be the first line on your page and correlate with the file name
- **✗ Don't** try to format markdown documents by using HTML tags or other means. The idea of markdown is to contain content only marked up with the content type (normal paragraph, headline, code, bullet list, highlighted text, ...) and let the renderer do the formatting for a consistent output.
- **✗ Don't** casually introduce a new style / language because you have a better idea. If you must, update similar nodes to follow the same style, or reach out to the owners of that code. But look for team agreement first.

API Documentation

DocFX can understand the source code structure and extract [API documentation](#) pages per member or type, enriched with information contained in the XML comments. That way code & API documentation is always in sync.

API Doc Extension (Overwrites)

Overwrites complement this with extended user documentation, directly written in prose Markdown. They contain further details, rich media and code examples via snippet references, which are automatically extracted from actual source code files being tested in the pipeline.

NOTE

UIDs create the link between an overwrite markdown file and the API node to merge the content in. Figuring out these UIDs is not trivial via the DocFX built-in features, but we're considering alternative tools to help here. Validating the correct use of UIDs and flag broken links - like in the case of API node renames - are also areas for custom tools to provide added value.

The following structure is followed for overwrites, which is blended into **Details** section of the API doc. All sections use level-5 (# #####) headlines for consistency with the rest of the generated doc:

Paragraph	Content	Test Method	Test Block	Service	Type
Test Technique	An introductory paragraph describing a typical use case this is intended for: - what are we testing on the device (background)? - what is a typical HW connection block diagram for testing using this test method (picture)? - what are typical value ranges and modes that are being programmed and what are the typical measured values expected for that typical block diagram? - what are things that can go wrong and cause unexpected values?	✓	?	✗	✗
Implementation	Summarize the implemented logic and sequence. Mention hierarchies if applicable.	✓	✓	?	?
Platform Specifics	Describe relevant information that is specific to some platforms only.	✓	✓	?	?
Pre Conditions	List any (unexpected) conditions that need to exist for successful operation.	✓	✓	?	?
Post Conditions	List any (unexpected) left-overs that are worth noting.	✓	✓	?	?
Limitations	List if and where the feature is deviating from the overall expectable use model.	✓	✓	?	?
Examples	Show typical use cases, add some meaningful context code applicable (not only a single line).	✗	✓	✗	✗
Code Reference	Reference the actual source code.	✓	?	✗	✗

Output Publishing

The output is published to a dynamic web page, nicely formatted and with cross-reference links automatically applied. Rich Search is available. All pages support direct editing, by clicking on the pen symbol users can apply changes right on the web page and submit a pull request with the chances to the repo maintainers.

Additionally, static HTML is produced to be included in the release package shipped to users.

Alternatives Considered

MyInfo

It is not currently planned to directly incorporate the C#RA documentation into MyInfo. A process to support automatic API documentation from XML comments is planned, but not yet available. Manually generating the documentation and keep it in sync with the code would result in significant overhead, and the authoring tools are not broadly accessible to application engineers.

However, since the focus lies on the content and that's where the vast majority of the effort is spent, it is expected that this can be easily transferred to MyInfo if and when that is needed.

Additional XML Tags

The use of `<exception>` tag was considered as it would be supported by VisualStudio IntelliSense. Since the `AlertService` centralizes the exception handling, no exceptions are raised from the code directly. That'll get in the way of the triple-slash `///` comment wizard automatically listing any exception in the method. Without that and no compiler warning either, it would be unlikely for the code base to end up with a consistent and complete way to list exceptions. Instead, exceptional cases would be listed in extended documentation along with other limitations or constraints.

The use of advanced XML tags right in the source code was considered as alternative to prose Markdown documentation being merged via overwrites. XML doc supports a wide range of additional tags like `<remarks>`, `<example>`, `<see>`, `<seealso>`, `<code>`, ... which however significantly bloat the source code itself. Because the product is being shipped as source code, and is intended for users to be accessible and intuitive, it was decided to limit XML tag use only to those that are directly reflected in IntelliSense.

Limited formatting options and access to media like images and animations and the lack of a snippet reference mechanism from verified code were additional counter-arguments.

Extensibility

The C#RA is delivered as source code, giving customers full access to the implementation. While this enables direct usage and deep customization, we strongly discourage modifying the core source. Custom changes can make future updates and upgrades difficult or incompatible. To ensure smooth upgrades and long-term maintainability, always prefer supported extensibility mechanisms—such as Extension Methods—over direct source modification.

Supported Extensibility: Extension Methods

Currently, C#RA supports extensibility primarily through **Extension Methods**. This .NET feature allows users to add new TestBlocks to interfaces from outside the original codebase, without modifying the core source. Extension Methods enable customers to name and implement their own methods, linking them to C#RA interfaces as needed.

NOTE

Inheritance, pre/post hooks, or other interception mechanisms are **not** supported in the current architecture. Extension Methods are the only officially supported way to extend functionality.

How Extension Methods Work

Extension Methods let customers implement additional functionality in their own projects. These methods are discoverable and callable as if they were part of the original interface or class, but ownership and responsibility for these methods remain with the customer.

TIP

For more details, see the official Microsoft documentation on [Extension Methods](#).

Example:

Adding an Extension Method to the interface `ILib.ISetup.IDc` with two arguments.

```
using Csra.Interfaces;

namespace Demo_CSRA {
    public static class CustomerExtensions {
        public static void CustomerExtension(this ILib.ISetup.IDc dc, string argument1, int
argument2) {
            // Do what you want
        }
    }
}
```

```
        }
    }
}
```

Usage:

The ExtensionMethod can be used like this.

```
[TestMethod]
public void UseExtensionMethod() {
    TheLib.Setup.Dc.CustomerExtension("firstArgument", 2);
}
```

This approach allows customers to extend the `TheLib` node with their own `TestBlocks`.

⚠️ WARNING

Extension Methods can only **add** functionality. If a future C#RA release introduces a method with the same signature, the built-in method will take precedence and **your extension will be ignored**.

Considerations and Limitations

- **Ownership:** Extension Methods are fully owned and maintained by the customer. C#RA does not enforce any naming or implementation rules for these methods.
- **Instrument Agnosticism:** Since extensions are implemented externally, they may not follow the same instrument-agnostic principles as the core C#RA.
- **No Pre/Post Hooks or Inheritance:** There is no built-in support for pre/post hooks, interception, or inheritance-based extension. All extensibility must be done via Extension Methods.

Customizing TestMethods

For more significant changes, such as altering arguments or logic in `TestMethods`, the recommended approach is to copy the relevant code into the customer project and modify as needed.

```
private Site<bool> _patResult;
private PatternInfo _patternInfoA;
private PatternInfo _patternInfoB;

[TestMethod, Steppable, CustomValidation]
public void Baseline(Pattern patternA, Pattern patternB, string setup = "") {

    if (TheExec.Flow.IsValidating) {
```

```

    _patternInfoA = new PatternInfo(patternA, true);
    _patternInfoB = new PatternInfo(patternB, true);
}

if (ShouldRunPreBody) {
    TheLib.Setup.LevelsAndTiming.Apply(true);
    Services.Setup.Apply(setup);
}

if (ShouldRunBody) {
    TheLib.Execute.Digital.RunPattern(_patternInfoA);
    TheLib.Execute.Digital.RunPattern(_patternInfoB);
    _patResult = TheLib.Acquire.Digital.PatternResults();
}

if (ShouldRunPostBody) {
    TheLib.Datalog.TestFunctional(_patResult, patternA);
}
}

```

Here, for example, a second **Pattern** argument was added and executed in sequence.

Summary

C#RA is designed to be used as-is or extended via supported .NET Extension Methods. Direct modification of the core source is discouraged to ensure upgradability. Extension Methods provide flexibility, but customers are responsible for their implementation and maintenance.

External Libraries

Static Dependencies

- IG-XL (sic!)
- hopefully nothing else

Conditional Dependencies

Conditional Project Reference

In scenarios where PortBridge is installed and licensed, it's dll must be referenced by the test code projects making use of it. Typically, such a reference is a hard dependency, and the project can not even build without it - even if it was not directly used.

On the other hand, a configuration without PortBridge will typically not have the software installed. There is no dll that could be referenced to satisfy the compiler.

To address both use cases, a conditional project reference is added:

```
<PropertyGroup>
    <!-- Check if external config or marker file exists -->
    <HuhuExists Condition="Exists('..\huhu.txt')">true</HuhuExists>
</PropertyGroup>
```

For this to work, new SDK project files are required. First, a custom MSBuild property `HuhuExists` is defined. It's set to true if a file `..\huhu.txt` exists. The relative path is from the location of the `.csproj` file, so in typical scenarios it would be next to the `.sln` file.

Since that custom property would be undefined in case the file does not exist, it's being set to `false` in that case:

```
<PropertyGroup>
    <!-- Default value if the flag file doesn't exist -->
    <HuhuExists Condition=" '$(HuhuExists)' == '' ">false</HuhuExists>
</PropertyGroup>
```

Now, the flag can be used as a condition to establish the project reference:

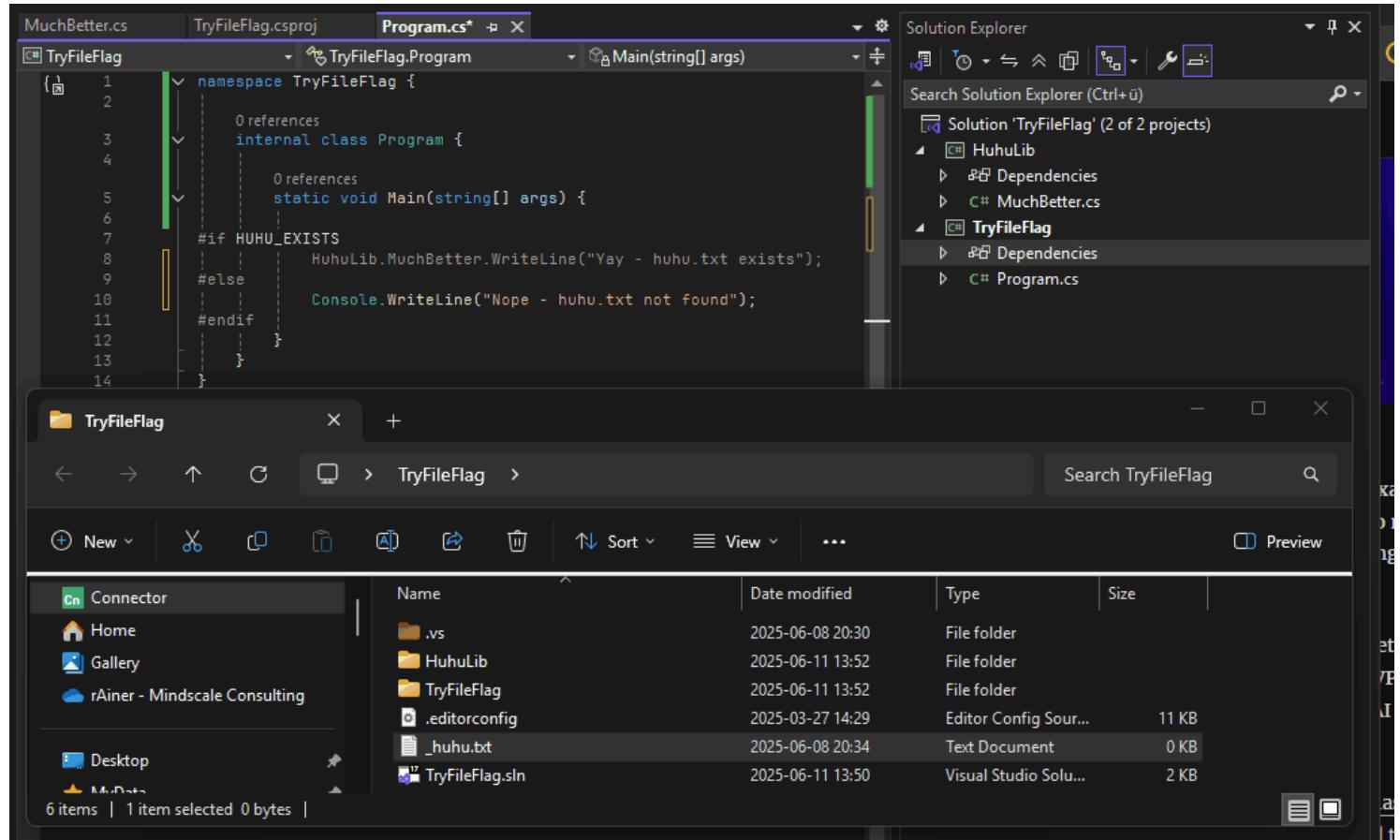
```
<ItemGroup Condition=" '$(HuhuExists)' == 'true' ">
    <ProjectReference Include=".\\HuhuLib\\HuhuLib.csproj" />
</ItemGroup>
```

Conditional Calls to PortBridge API

The same mechanism can be used for conditional calls to the PortBridge API. For that to work, a compiler constant is defined:

```
<PropertyGroup Condition=" '$(HuhuExists)' == 'true' ">
    <DefineConstants>$(DefineConstants);HUHU_EXISTS</DefineConstants>
</PropertyGroup>
```

With that, code can be conditionally enabled / disabled for the build process:



Specific APIs are fully accessible including intellisense and type-ahead when that is intended, and alternative implementations are used when that isn't available.

Risks

The potential need for substantial code sections using `#if` compiler directives, and the mechanics involved with the conditional project references will add complexity to the code base for this option. The admittedly high risk of overburdening maintenance cost will be mitigated by careful choices and close monitoring of the actual implementation.

Alternatives Considered

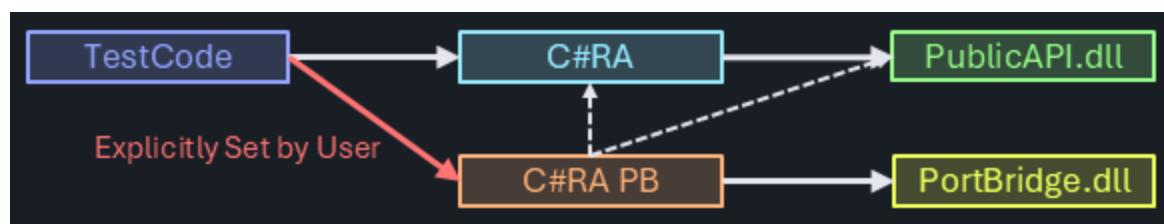
The agreement on the proposed model above was the result of controversial discussions and significant design work in alternative approaches. Either model has a significant impact to the project and comes with tough implications, some of which are hard to fully foresee in the current project state. The eventual choice was made because the proposed model seemed to have the least limitations for future project enhancements. Prototyping of the alternatives could not fully confirm feasibility of either.

Should these assumptions turn out incorrect, the project may switch to one of the following options, if they end up better serving the needs.

Dedicated Extension Project

The conditional project dependencies for external libraries could be handled by a separate, project specific (like C#RA_XYZ) extension project, which statically references the library dll. In the extension project, all library specific functionalities are defined and blended into the regular C#RA use model.

The user opts into Library use by explicitly setting a project reference to the C#RA Library extension project.



Internally, the C#RA library extension project follows the same structure as the main project, it only creates a logical separation for library specific language. It is integrated into the release package for all users. It's limited to separate language for the extension feature, possibly also substituting functionality of the base class. It wouldn't however allow seamlessly blending additional coverage into the existing use model.

Common Interfaces / Factory Pattern / Dynamically Loaded References

Alternative (Library / no Library) implementations could also be achieved with a common wrapper interface class, and using a factory pattern to instantiate the relevant feature at runtime. The following downsides were captured - and led to deciding against this model:

- runtime vs. design time linking: intellisense, type-ahead of the original library implementation can't be used as the project isn't known at design time.
- additional abstraction layer: Library programming would have to go through a common interface, disguising the originally intended use model for the feature. Given the Library is very much in development, this additional layer was considered a big risk.
- common interface would be more restrictive in offering custom functionality of either option

Questions

- How to handle different versions of library dependencies?
- How to support multiple versions? Customer A needs 1, customer B needs 2?
- Does that mean we add another dimension (i.e., duplicate) to all unit and integration testing efforts for every library added? Likely yes ...

Instrument Specific Features

The C# reference architecture introduces a unified conceptual model for managing data associated with one or multiple conditions applicable to pins and pin groups. This methodological approach facilitates a level of abstraction comparable to that found in single-condition scenarios, thereby contributing to process optimization and simplifying user interaction with the system. Eliminating friction and redundancies from the workflow is a key objective of this architecture, ensuring an efficient and intuitive framework for application development.

However, in practice, the need arises to implement specific settings for certain instruments, where configuration parameters can no longer be considered common values across all instances of the process. This diversity in technical requirements may lead to increased complexity in function structure and an overload of user calls. As a result, it is necessary to develop a solution that systematically and intelligibly manages these variables.

The proposed concept aims to address this challenge through a well-structured methodology that allows the definition and application of non-uniform configuration parameters in an intuitive and efficient manner. The implementation of a coherent mechanism for managing these variables contributes to error reduction, improved maintenance, and increased accessibility in the use of instruments integrated into the C# reference architecture. This approach not only optimizes technical processes but also strengthens the system's flexibility and scalability, providing developers with a robust framework for managing data in variable conditions.

The Concept of Implementation

The implementation of the concept requires a structured approach to ensure that parameterization is managed efficiently and in a scalable manner. This involves defining a dedicated class that contains all the necessary parameters for the process, ensuring a clear separation between configuration logic and test execution. In this paradigm, parameter invocation is performed via an instantiated object of the class, utilized during the validation phase of the test block.

This methodology offers several fundamental advantages. Firstly, it ensures data integrity and organization in a coherent manner, preventing uncontrolled access to internal settings and reducing the risk of errors during implementation. Secondly, the proposed solution facilitates the expansion and adaptability of the architecture, allowing the addition of new parameters without disrupting the existing structure. This aspect is essential for maintaining system flexibility, particularly in scenarios where configuration requirements evolve with the integration of new instruments.

Through this approach, the goal is to optimize user interaction with the system and reduce operational complexity, providing a robust framework for developing scalable solutions that can be easily integrated into the existing architecture. Thus, the implementation of the concept enhances the organization and

management of parameters from the perspective of the future development of the technical infrastructure.

User Interface

The analysis of settings and their applicability represents a fundamental element in optimizing the interaction between the user and the system. In this context, the table presented below serves to provide a clear correlation between the parameters existing in the user interface and the configurations applied at the instrument level, thus ensuring precise alignment with the initialization process of each variable.

A systematic approach to these settings not only enables efficient utilization of available resources but also standardizes how variables are manipulated and integrated into operational processes. By identifying the relationships between configuration parameters and their impact on the functioning of the instruments, a robust optimization methodology can be developed, thereby reducing error risks and enhancing system reliability.

Therefore, the structure of the presented table not only offers a concise description but also serves as a reference point for users in the configuration process, contributing to a clear and coherent experience.

Table for Setup.Dc

Property	Type	PPMU	DCVI	DCVS	Purpose	Observations
Gate	bool	✓	✓	✓	Sets the gate for the specified pins.	
Mode	TLibOutputMode	✓	✓	✓	Sets the operating mode for the specified pins.	
Voltage	double	✓	✓	✓	Sets the output voltage for the specified pins.	
VoltageAlt	double			✓	Sets the alternate output voltage for	

Property	Type	PPMU	DCVI	DCVS	Purpose	Observations
					the specified pins.	
Current	double	✓	✓	✓	Sets the output current for the specified pins.	For DCVS, the 'current' option includes both source and sink for Foldlimit.
VoltageRange	double		✓	✓	Sets the voltage range for the specified pins.	
CurrentRange	double	✓	✓	✓	Sets the current range for the specified pins.	
ForceBandwidth	double		✓	✓	Sets the output compensation bandwidth for the specified pins.	
MeterMode	Measure	✓	✓	✓	Sets the meter mode for the specified pins.	
MeterVoltageRange	double		✓	✓	Sets the meter voltage range for the specified pins.	
MeterCurrentRange	double	✓	✓	✓	Sets the meter current range for the specified pins.	

Property	Type	PPMU	DCVI	DCVS	Purpose	Observations
MeterBandwidth	double		✓	✓	Sets the meter filter for the specified pins.	
SourceFoldLimit	double			✓	Sets the source fold limit for the specified pins.	
SinkFoldLimit	double			✓	Sets the sink fold limit for the specified pins.	
SourceOverloadLimit	double			✓	Sets the source overload limit for the specified pins.	
SinkOverloadLimit	double			✓	Sets the sink overload limit for the specified pins.	
VoltageAltOutput	bool			✓	Sets the output DAC used to force voltage (true for alternate or false for main).	
BleederResistor	bool		✓		Sets the bleeder resistor's connection state for the specified pins.	

Property	Type	PPMU	DCVI	DCVS	Purpose	Observations
ComplianceBoth	double		✓		Sets both compliance ranges for the specified pins.	
CompliancePositive	double		✓		Sets the positive compliance range for the specified pins.	Both can be set through the variable 'ComplianceBoth'.
ComplianceNegative	double		✓		Sets the negative compliance range for the specified pins.	
ClampHiV	double	✓			Sets the high voltage clamp value for the specified pins.	
ClampLoV	double	✓			Sets the low voltage clamp value for the specified pins.	
HighAccuracy	bool	✓			Sets the enabled state of the high accuracy measure voltage.	
SettlingTime	double	✓			Sets the required additional settling time for the high accuracy	

Property	Type	PPMU	DCVI	DCVS	Purpose	Observations
					measure voltage mode.	
HardwareAverage	double		✓		Sets the meter hardware average value for the specified pins.	

Table for Setup.Digital

Table for Setup.Digital.ModifyPins()

Property	Type	Support	Purpose	Observations	API Endpoint
AlarmType	tlHSDMAlarm	✓	Sets the alarm type for the specified pins.	Should be set together with AlarmBehavior	TheHdw.Digital
AlarmBehavior	tlAlarmBehavior	✓	Sets the alarm behavior for the specified pins.	Should be set together with AlarmType	TheHdw.Digital
DisableCompare	bool	✓	Disables the comparators for the specified pins		TheHdw.Digital

Property	Type	Support	Purpose	Observations	API Endpoint
DisableDrive	bool	✓	Disables the drivers for the specified pins		TheHdw.Digital
InitState	ChInitState	✓	Sets the initial state of the pins		TheHdw.Digital
StartState	ChStartState	✓	Sets the start state of the pins		TheHdw.Digital
CalibrationExcluded	bool	✓	Sets the specified pins to be excluded from job dependent calibration		TheHdw.Digital
CalibrationHighAccuracy	bool	✓	Enables or disables calibration high accuracy mode for the specified pins		TheHdw.Digital

Table for Setup.Digital.ModifyPinsLevels()

Property	Type	Support	Purpose	Observations
DifferentialLevelsType	ChDiffPinLevel	✓	Sets the differential levels type for the	Should be set to DifferentialLevelsV

Property	Type	Support	Purpose	Observations
			specified pins	
DifferentialLevelsValue	double	✓	Sets the specified differential pin level type for the specified pins	Should be set toge DifferentialLevelsT
DifferentialLevelsValuesType	==TLibDiffLvlValType[]==	✓	Sets the differential levels values type for the specified pins	Should be set toge DifferentialLevelsV
DifferentialLevelsValues	double[]	✓	Sets the specified differential pin levels values type for the specified pins	Should be set toge DifferentialLevelsV
LevelsDriverMode	tIDriverMode	✓	Sets the driver mode for the specified pins	
LevelsType	ChPinLevel	✓	Sets the level type	

Property	Type	Support	Purpose	Observations
			for the specified pins	
LevelsValue	double	✓	Sets the value for the specified level type on the specified pins	Should be set to the LevelsType
LevelsValuePerSite	SiteDouble	✓	Sets the value for the specified level type for the specified pins on each site	Should be set to the LevelsType
LevelsValues	PinListData	✓	Sets the value for the specified level value for each specified site and each specified pin	Should be set to the LevelsType

Table for Setup.Digital.ModifyPinsTiming()

Property	Type	Support	Purpose	Observations
TimingClockOffset	double	✓	Sets the offset value between a DQS bus and a DUT clock in a DDR Protocol Aware test program for the specified pins	
TimingClockPeriod	double	✓	Sets the current value for the period for the specified clock pins	
TimingDisableAllEdges	bool	✓	Disables all edges (drive and compare) for the specified pins	
TimingEdgeSet	string	✓	Sets the edgeset name for the specified pins	
TimingEdgeVal	chEdge	✓	Sets the timing edge	

Property	Type	Support	Purpose	Observations
			for the specified pins	
TimingEdgeEnabled	bool	✓	Sets the enabled state for the specified pins and timing edge	Should be set to TimingEdgeEdgeS TimingEdgeVal
TimingEdgeTime	double	✓	Sets the edge value for the specified pins and timing edge	Should be set to TimingEdgeEdgeS TimingEdgeVal
TimingRefOffset	double	✓	Sets the offset value between the specified source synchronous reference (clock) pin and its data pins	
TimingSetup1xDiagnosticCapture	string	✓	Sets up special dual-bit diagnostic capture in CMEM fail capture (LFVM) memory using the 1X pin setup	

Property	Type	Support	Purpose	Observations
			for the specified pins and Time Sets sheet name	
TimingSrcSyncDataDelay	double	✓	Sets the strobe reference data delay for individual source synchronous data pins	
TimingOffsetType	tOffsetType	✓	Sets the timing offset type for the specified pins	
TimingOffsetValue	double	✓	Sets the timing offset value for the specified pins	Should be set to <code>TimingOffsetType</code>
TimingOffsetEnabled	bool	✓	Sets the timing offset enabled state for the specified pins	Should be set to <code>TimingOffsetType</code>
TimingOffsetSelectedPerSite	SiteLong	✓	Sets the active offset	Should be set to <code>TimingOffsetType</code>

Property	Type	Support	Purpose	Observations
			index value for the specified pins on each site	
TimingOffsetValuePerSiteIndex	int	✓	Set the timing offset index value. The valid index range is 0-7	Should be set to <code>TimingOffsetType</code> <code>TimingOffsetValue</code>
TimingOffsetValuePerSiteValue	SiteDouble	✓	Sets the current value for the offset at a specific index location that is to be applied to the timing values for the specified pins on each site	Should be set to <code>TimingOffsetType</code> <code>TimingOffsetValue</code>
AutoStrobeEnabled	AutoStrobeEnableSel	✓	Enable state of the AutoStrobe engine for the specified pins	
AutoStrobeNumSteps	int	✓	Sets the number of steps on the	

Property	Type	Support	Purpose	Observations
			AutoStrobe engines for the specified pins	
AutoStrobeSamplesPerStep	int	✓	Sets the number of samples per step on the AutoStrobe engines for the specified pins	
AutoStrobeStartTime	double	✓	Sets the start time on the AutoStrobe engines for the specified pins	
AutoStrobeStepTime	double	✓	Sets the step time on the AutoStrobe engines for the specified pins	
FreeRunningClockEnabled	bool	✓	Sets the enable state of the free-running clock for the	

Property	Type	Support	Purpose	Observations
			specified pins	
FreeRunningClockFrequency	double	✓	Sets the frequency of the free-running clock for the specified pins	
FreqCtrEnable	FreqCtrEnableSel	✓	Sets the frequency counter's enable state for the specified pins	
FreqCtrEventSlope	FreqCtrEventSlopeSel	✓	Sets the frequency counter's event slope for the specified pins	
FreqCtrEventSource	FreqCtrEventSrcSel	✓	Sets the frequency counter's event source for the specified pins	
FreqCtrInterval	double	✓	Sets the duration of time to capture the	

Property	Type	Support	Purpose	Observations
			frequency counter data for the specified pins	

Not Offered for Setup.Digital.ModifyPins

Irrelevant commands are filtered out based on these criteria:

- The API must be compatible with **UltraPin2200**.
- The API must fit the "setup" definition (like read-only ones, which more like "acquire").
- The API cannot be no return value actions like `.Start()`, `.Stop()`, `.Save()`, `.Restore()` and so on ("execution" commands).
- The API commands' parameters are too complex, cannot support currently, like pattern data modify commands.

Table for not offered for Setup.Digital.ModifyPins

Property	Observations	API Endpoint
ClearFail	Action command	TheHdw.Digital.Pins(PinList)
Connect	Action command	TheHdw.Digital.Pins(PinList)
Connected	Read-only	TheHdw.Digital.Pins(PinList)
Disconnect	Action command	TheHdw.Digital.Pins(PinList)
FailCount	Read-only	TheHdw.Digital.Pins(PinList)
FailCountLimit	Read-only	TheHdw.Digital.Pins(PinList)
FailCountLimitReached	Read-only	TheHdw.Digital.Pins(PinList)
FailCountOnly	Read-only	TheHdw.Digital.Pins(PinList)
Failed	Read-only	TheHdw.Digital.Pins(PinList)
FailedPerSiteArray	Read-only	TheHdw.Digital.Pins(PinList)

Property	Observations	API Endpoint
GetFailCountArray	Read-only	TheHdw.Digital.Pins(PinList)
LockState	used only on the UltraPin4000 and HPM	TheHdw.Digital.Pins(PinList)
put_FailCountLimit	used only on the UltraPin4000 and HPM	TheHdw.Digital.Pins(PinList)
FindEdge	Read-only	TheHdw.Digital.Pins(PinList).AutoStrobe
Exclude	"Avoid using this method when creating test programs using the UltraPin2200 on an UltraFLEXplus." -- MyInfo	TheHdw.Digital.Pins(PinList).Calibration
Trace	"Avoid using this method when creating test programs using the UltraPin2200 on an UltraFLEXplus." -- MyInfo	TheHdw.Digital.Pins(PinList).Calibration.DIB
Data	Read-only	TheHdw.Digital.Pins(PinList).CMEM
FailIndexList	Read-only	TheHdw.Digital.Pins(PinList).CMEM
ModuleCycleData	Read-only	TheHdw.Digital.Pins(PinList).CMEM
StoredCycleData	Read-only	TheHdw.Digital.Pins(PinList).CMEM

Property	Observations	API Endpoint
StoredFailCount	Read-only	TheHdw.Digital.Pins(PinList).CMEM
PeakingMode	used only on the UltraPin4000 and HPM	TheHdw.Digital.Pins(PinList).DifferentialLevels
Restore	Action command	TheHdw.Digital.Pins(PinList).DifferentialLevels
Save	Action command	TheHdw.Digital.Pins(PinList).DifferentialLevels
TerminationMode	used only on the UltraPin4000 and HPM	TheHdw.Digital.Pins(PinList).DifferentialLevels
IsRunning	Read-only	TheHdw.Digital.Pins(PinList).FreeRunningClock
Start	Action command	TheHdw.Digital.Pins(PinList).FreeRunningClock
Stop	Action command	TheHdw.Digital.Pins(PinList).FreeRunningClock
Clear	Action command	TheHdw.Digital.Pins(PinList).FreqCtr
MeasureFrequency	Read-only	TheHdw.Digital.Pins(PinList).FreqCtr
Read	Read-only	TheHdw.Digital.Pins(PinList).FreqCtr
SetupFreqCounter	used only on the UltraPin1600	TheHdw.Digital.Pins(PinList).FreqCtr
Start	Action command	TheHdw.Digital.Pins(PinList).FreqCtr
CapturedFailCycleInfo	Read-only	TheHdw.Digital.Pins(PinList).HRAM

Property	Observations	API Endpoint
PatData	Read-only	TheHdw.Digital.Pins(PinList).HRAM
PinData	Read-only	TheHdw.Digital.Pins(PinList).HRAM
PinPF	Read-only	TheHdw.Digital.Pins(PinList).HRAM
ReadDataBits	Read-only	TheHdw.Digital.Pins(PinList).HRAM
ReadDataWord	Read-only	TheHdw.Digital.Pins(PinList).HRAM
Move	not used with the UltraPin1600 and UltraPin2200	TheHdw.Digital.Pins(PinList).Jitter
DifferentialModeEnabled	Read-only	TheHdw.Digital.Pins(PinList).Levels
PeakingMode	used only on the UltraPin4000 and HPM	TheHdw.Digital.Pins(PinList).Levels
Restore	Action command	TheHdw.Digital.Pins(PinList).Levels
Save	Action command	TheHdw.Digital.Pins(PinList).Levels
TerminationMode	used only on the UltraPin4000 and HPM	TheHdw.Digital.Pins(PinList).Levels
GetVectorData	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName)
GetVectorScanData	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName)

Property	Observations	API Endpoint
GetVectorState	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName)
ModifyVectorBlockData	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName)
ModifyVectorBlockDataNSite	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName)
ModifyVectorData	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName)
ModifyVectorDataNSite	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName)
ModifyVectorScanData	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName)
ModifyVectorScanDataNSite	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName)
SetDataWords	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName)
SetDataWordsPerSite	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName)
SetVectorData	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName)

Property	Observations	API Endpoint
SetVectorScanData	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName)
SetVectorState	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName)
AllocateScanOffset	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName).NonCc
AllocateVectorOffset	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName).NonCc
Deallocate	Action command	TheHdw.Digital.Pins(PinList).Patterns(PatName).NonCc
IsAllocated	Read-only	TheHdw.Digital.Pins(PinList).Patterns(PatName).NonCc
ModifyScanData	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName).NonCc
ModifyVectorData	Too complex, not support currently	TheHdw.Digital.Pins(PinList).Patterns(PatName).NonCc
DataPins	Read-only	TheHdw.Digital.Pins(PinList).SourceSync
PinType	Read-only	TheHdw.Digital.Pins(PinList).SourceSync
ReferencePin	Read-only	TheHdw.Digital.Pins(PinList).SourceSync
EdgeSet	Read-only	TheHdw.Digital.Pins(PinList).Timing
LateExpect	used only on the HPM	TheHdw.Digital.Pins(PinList).Timing
RestoreCMEMFailCaptureMap	Action command	TheHdw.Digital.Pins(PinList).Timing

Property	Observations	API Endpoint
StrobeRefSetupName	Read-only	TheHdw.Digital.Pins(PinList).Timing
Amplitude	used only on the HPM	TheHdw.Digital.Pins(PinList).Timing.Jitter.Insertion
Mode	used only on the HPM	TheHdw.Digital.Pins(PinList).Timing.Jitter.Insertion
Period	used only on the HPM	TheHdw.Digital.Pins(PinList).Timing.Jitter.Insertion
Max	Read-only	TheHdw.Digital.Pins(PinList).Timing.Offset
Min	Read-only	TheHdw.Digital.Pins(PinList).Timing.Offset
DefineSetup	used only on the UltraPin4000 and HPM	TheHdw.Digital.Pins.Tracker
Offset	used only on the UltraPin4000 and HPM	TheHdw.Digital.Pins.Tracker
Reset	used only on the UltraPin4000 and HPM	TheHdw.Digital.Pins.Tracker
Status	used only on the UltraPin4000 and HPM	TheHdw.Digital.Pins.Tracker
Data	used only on the UltraPin4000 and HPM	TheHdw.Digital.Pins(PinList).Tracker.History

Property	Observations	API Endpoint
MaxVal	used only on the UltraPin4000 and HPM	TheHdw.Digital.Pins(PinList).Tracker.History
MinVal	used only on the UltraPin4000 and HPM	TheHdw.Digital.Pins(PinList).Tracker.History

From the user's perspective, the instantiation process is optimized to include only the necessary settings, thereby avoiding system overload with unnecessary parameters. This approach not only improves resource management but also facilitates a more intuitive interaction with the infrastructure. After instantiation, the parameter is transferred via the function, ensuring a clear and coherent organization of the process.

Example of the user interface

```

private DcParameters _modifySettings;

[TestMethod, Steppable, CustomValidation]
public void Baseline(PinList pinList, ..., double clampHi, double clampLo, double
bandwidthSetting) {

    if (TheExec.Flow.IsValidating) {
        _modifySettings = new DcParameters() {
            ClampHiV = clampHi,
            ClampLoV = clampLo,
            ForceBandwidth = bandwidthSetting
        };
    }

    if (ShouldRunBody) {
        TheLib.Setup.Dc.Modify(_pins, _modifySettings);
    }
}

private DigitalParameters _digModifySettings;

[TestMethod, Steppable, CustomValidation]
public void Baseline(PinList pinList, ..., bool disableDrive, ChInitState initState,

```

```

ChStartState startState) {

    if (TheExec.Flow.IsValidating) {
        _digModifySettings = new DigitalParameters() {
            disableDrive = false,
            initState = ChInitState.Hi,
            startState = ChStartState.Hi
        };
    }

    if (ShouldRunBody) {
        TheLib.Setup.Digital.Modify(_pins, _digModifySettings);
    }
}

```

Implementation

In this context, parameter definition is carried out within a dedicated class that serves as a container for the settings required by each individual instrument. This approach enables a clear separation between data retrieval and configuration logic or test execution functionality, thereby contributing to an optimized process management.

In the context of calling method `Modify`, the code execution process involves verifying the specific setting associated with the instrument, thereby ensuring strict compliance with predefined configuration requirements. This verification serves as a control mechanism, determining whether the necessary parameter has been defined to allow its modification within the instrument's functionality.

```

// Setup.Dc
internal static void Modify(Pins pins, DcParameters parameters) {
    if (pins.ContainsFeature(InstrumentFeature.Ppmu, out string ppmuPins)) {
        ModifyPpmu(ppmuPins, parameters);
    }
    if (pins.ContainsFeature(InstrumentFeature.Dcv, out string dcviPins)) {
        ModifyDcv(dcvPins, parameters);
    }
    if (pins.ContainsFeature(InstrumentFeature.DcvS, out string dcvsPins)) {
        ModifyDcvS(dcvsPins, parameters);
    }
}

private static void ModifyPpmu(string pins, DcParameters ppmuParameters) {
    if (ppmuParameters.ClampHiV.HasValue) ppmu.ClampVHi.Value
= ppmuParameters.ClampHiV.Value;
    if (ppmuParameters.ClampLoV.HasValue) ppmu.ClampVLo.Value

```

```

= ppmuParameters.ClampLoV.Value;
}

private static void ModifyDcvi(string pins, DcParameters dcviParameters) {
    if (dcviParameters.ForceBandwidth.HasValue) dcvi.NominalBandwidth.Value =
dcviParameters.ForceBandwidth.Value;
}

private static void ModifyDcvs(string pins, DcParameters dcvsParameters) {
    if (dcvsParameters.ForceBandwidth.HasValue) dcvs.BandwidthSetting.Value =
dcvsParameters.ForceBandwidth.Value;
}

// Setup.Digital
internal static void Modify(Pins pins, DigitalParameters parameters) =>
TheLib.Setup.Digital.Modify(pins,...,
    parameters.disableDrive, parameters.initState, parameters.startState);
internal static void Modify(Pins pins,..., bool disableDrive, ChInitState initState,
ChStartState startState) {
    if (pins.ContainsFeature(InstrumentFeature.Digital, out string digitalPins)) {
        if (initState.HasValue) TheHdw.Digital.Pins(digitalPins).InitialState
= initState.Value;
        if (startState.HasValue) TheHdw.Digital.Pins(digitalPins).StartState
= startState.Value;
    }
}

```

Templates

Templates (Functional_T, PPMU_T, Empty_T, ...) have been part of IG-XL from day 1 on. Their best days are over, but they still have a loyal user base, who already repeatedly asked for .NET versions of them. They serve these aspects:

- provide canned functionality for common test scenarios
- reflect (what used to be) recommended practice
- dramatically short-cut test development: fill out a form
- allow customization

There's a great overlap with the high level goals of the C# Reference Architecture, and reproducing their exact functionality with it is an important check step. This isn't questioning how reasonable the originally defined scope (too many parameters? Too generic?) is, but rather proof-of-concept that the essential components are available and working together.

Aside from the exact representations of the legacy templates, more streamlined and modern ones are offered. Users **could** use the original ones in .NET if they wanted, but they could also decide for better ones that are offered in addition.

Feature Modularity

In order to avoid overloaded and bulky implementations, an opt-in approach is chosen for features where possible. Have our templates in `Template.xla` already been unhandy, imagine how they would look like if all the power of .NET would also be crammed in there.

To allow for low entry barrier and to let users decide which features they need and want, a modular approach is followed. The exact model varies for the different features, but in general, users should be able to mix and match.

The idea is explained at the example of a simple test method, opting in to debug and argument validation functionality.

A Simple Test Method

The following code shows a minimum implementation of a basic continuity test. The [test abstraction](#) methodology is used for concise code:

```
using static Demo.TestLib;

[TestMethod]
public void SimpleContinuity(string digPins, string powerPins, double forceCurrent) {

    // setup
    string allPins = Utils.MergePinLists(digPins, powerPins);
    Connect(allPins);
    Setup.ForceV(powerPins, 0 * V);
    Setup.ForceI(digPins, forceCurrent, Measure.Voltage, 2 * V, forceCurrent);
    Setup.Gate(allPins, true);

    // measure
    TheHdw.Wait(1 * ms);
    PinSite<double> meas = Acquire.ReadMeter(digPins);

    // reset
    Setup.Gate(allPins, false);
    DisConnect(allPins);

    // datalog
    Datalog.TestParametric(meas, "V");
}
```

Example: Add Pre- / Body / Post Stepping Capability

If desired, the concept of steppable test methods can be easily added. The code would only require slight modifications:

```
[TestMethod, Steppable]
public void SimpleContinuityWithStepping(string digPins, string powerPins, double forceCurrent) {

    string allPins = Utils.MergePinLists(digPins, powerPins);
    PinSite<double> meas = null;

    if (ShouldRunPreBody) PreBody();
    if (ShouldRunBody) Body();
    if (ShouldRunPostBody) PostBody();

    void PreBody() {
        Connect(allPins);
        Setup.ForceV(powerPins, 0 * V);
        Setup.ForceI(digPins, forceCurrent, Measure.Voltage, 2 * V, forceCurrent);
        Setup.Gate(allPins, true);
    }

    void Body() {
        TheHdw.Wait(1 * ms);
        meas = Acquire.ReadMeter(digPins);
    }

    void PostBody() {
        Setup.Gate(allPins, false);
        DisConnect(allPins);
        Datalog.TestParametric(meas, "V");
    }
}
```

The stepping feature allows compliant test instances to have three distinct parts, which can be individually stepped through with Flow Breakpoints. The code for Three local methods are added, with each covering the code parts specific for the individual steps. Called from the flow controller 3 consecutive times, this test method branches into the corresponding Pre / Body / Post part via a central **if** structure at the top.

(i) TIP

[Local functions](#) have access to the arguments and local variables of the hosting member. It's not needed to hand these through via arguments. The compiler takes care of that, resulting in a cleaner code with less errors.

Example: Add Argument Validation Capability

To offer more convenience to the users of this generic test method, it may offer argument validation. Also that functionality can be added in a modular fashion, leaving the rest of the code structure untouched.

```
[TestMethod, Steppable, CustomValidation]
public void SimpleContinuityWithValidation(string digPins, string powerPins, double forceCurrent) {

    string allPins = Utils.MergePinLists(digPins, powerPins);
    PinSite<double> meas = null;

    if (TheExec.Flow.IsValidating) ValidateArgs();
    if (ShouldRunPreBody()) PreBody();
    if (ShouldRunBody()) Body();
    if (ShouldRunPostBody()) PostBody();

    void ValidateArgs() {
        Validate.IsInRange(forceCurrent, -200 * uA, 200 * uA, $"Force Current too large.");
    }

    void PreBody() {
        Connect(allPins);
        Setup.ForceV(powerPins, 0 * V);
        Setup.ForceI(digPins, forceCurrent, Measure.Voltage, 2 * V, forceCurrent);
        Setup.Gate(allPins, true);
    }

    void Body() {
        TheHdw.Wait(1 * ms);
        meas = Acquire.ReadMeter(digPins);
    }

    void PostBody() {
        Setup.Gate(allPins, false);
        DisConnect(allPins);
    }
}
```

```
Datalog.TestParametric(meas, "V");  
}
```

The validation part is also encapsulated in a local function, called from the high-level selector part at the top.

Multi-Target Support for Different IG-XL Versions

As more internal and external users adopt the C#RA, limiting development to a single IG-XL version has become increasingly impractical:

- New IG-XL features requested for the C#RA project remain inaccessible unless the entire team upgrades to a newer version.
- Ideally, such features should be verified as soon as they're implemented—to provide early feedback to the designers and reduce technical debt promptly.
- Without official support for multiple IG-XL versions, everyone must migrate simultaneously:
 - All collaborators and contributors
 - All users who want to continue receiving updates
 - IG-XL Test Harness (which must match the version)
 - Self-hosted runners
 - Online testers

Since we can't always dictate when users switch IG-XL versions—or whether they switch at all—C#RA must support targeting multiple IG-XL versions concurrently.

Strategy

Official support for any version essentially means testing that it works as expected. While this shouldn't require significant extra functionality, a support claim without a verification proof is meaningless.

As a result, all supported IG-XL versions must receive equivalent test coverage. Although development may focus on a specific version, test automation should run across all supported versions, ideally in parallel to maintain short cycle times. The same applies to online testing and performance profiling to ensure full confidence.

Compiler Directives

Incompatible API changes require conditional compilation to ensure successful builds. The simplest solution is using compiler directives (#if) to enable or disable code blocks based on preprocessor symbols:

```
#if IGXL_10_60_01_uflx
    TheExec.AddOutput("important message", ColorConstants.vbRed, true);
#else
    TheExec.AddOutput("important message", ColorConstants.Red, true);
#endif
```

This structure enables the legacy syntax (`ColorConstants.vbRed`) for IG-XL 10.60.01 and uses the new one (`ColorConstants.Red`) for all other versions. Since IG-XL 10.60.01 is the only pre-11.00 version still in use,

this is a safe approach here. Other scenarios might require finer granularity for different versions.

IG-XL Version-Specific Preprocessor Symbols

IG-XL does not natively provide version-based preprocessor symbols in its test and DSP code templates, but this can be added manually in the `.csproj` files:

```
<!-- This section is added for multi-target support of IG-XL versions. It'll create a
preprocessor symbol to allow wrapping incompatible IG-XL code within #if directives -->
<PropertyGroup>
    <RawPath>$(IGXLROOT)</RawPath> <!-- Get the environment variable -->
</PropertyGroup>
<PropertyGroup Condition="'$(<RawPath>)' != ''"> <!-- Only run the symbol-building logic if
IGXLROOT is defined -->
    <LastSegment>$([System.IO.Path]::GetFileName($(<RawPath>)))</LastSegment> <!-- Get the
last segment of the path (e.g., 10.60.01_uflx) -->
    <CleanVersion>$([System.String]::Copy('$(<LastSegment>)').Replace('.', '_'))
</CleanVersion> <!-- Replace dots with underscores -->
    <SymbolName>IGXL_$(CleanVersion)</SymbolName> <!-- Construct the symbol name -->
    <DefineConstants>$(<DefineConstants>);$(<SymbolName>)</DefineConstants> <!-- Append the new
symbol to DefineConstants -->
</PropertyGroup>
```

This injects a version-specific symbol like `IGXL_10_60_01_uflx`, which can be used in `#if` directives. In case a symbol is not defined (= a different version is currently installed), the wrapped code is ignored.

Logical operators such as `&&`, `||`, and `!` can be used to build flexible branching logic.

If this approach proves reliable and beneficial, IG-XL may consider incorporating it into its templates. For now, all relevant C# project files in C#RA have been updated manually.

Cleaning Up Outdated Versions

As development shifts to newer IG-XL versions, support for outdated ones should be removed promptly. Eliminating obsolete code paths keeps the codebase clean, maintainable, and easier to navigate.

Unit Testing

Unit Testing with the IG-XL Test Harness is based on the idea that IG-XL does not need to be installed on the target machine. That however means that the `IGXLROOT` environment variable wouldn't be defined either.

To make the above mechanics usable in this scenario, a version compatible `IGXLROOT` environment variable must be created before launching the unit tests. Then, the same mechanism can be used to control version divergent code execution flows also in the tests.

Costs & Effort

The effort to implement and maintain multi-version support can be grouped into the following categories:

- Duplicate logic or implementation for version-specific features
- Increased friction during development and verification
- Maintenance of more complex code structures and cluttered language
- Setup of additional VDIs for alternate IG-XL versions for all team members
- Duplication of offline unit test runs
- Duplication of online regression test handling
- Version-specific performance and result analysis
- Structural documentation updates to support version differences

Many of these costs can be mitigated through automation and process improvements. It's an upfront investment in one milestone that quickly pays off with a smoother workflow moving forward.

IMPORTANT

Even with solid tooling and automation, every officially supported IG-XL version adds recurring cost and maintenance overhead. New versions should be added cautiously, and outdated versions dropped as soon as feasible.

Alternatives Considered

Separate Projects

Duplicating the entire code base for what is expected to only be marginal differences in a small number of places would be a gross violation of the single source of truth principle. The cost of maintenance and likelihood (certainty!) to introduce issues would be enormous.

Interface Abstraction

Using interface abstraction to isolate diverging APIs was considered. However, because of the size of the IG-XL PublicAPI and the relatively small diverging areas as well as the transient nature of most differences, this approach was deemed impractical.

Code Generation

Code generation to handle syntax differences would introduce high complexity and poor maintainability. It would also be difficult to encapsulate (hide) effectively from end users.

Pre-Build Scripting or File Swapping

This approach shares the same mechanics as `#if` directives but places the logic outside the source files. This separation would make it harder to track and maintain. Keeping the logic inline in source code ensures clarity and transparency.

Public API Version Symbols

IG-XL's Public API has its own versioning scheme, which always increments with each IG-XL release—regardless of language changes. While technically feasible, these versions are less familiar and intuitive for most users. Using them would increase the risk of mistakes.

However, users with specific needs can implement their own version-based symbols using the same `.csproj` mechanism shown above.

Persistent Data Storage

Test programs frequently have a need to store data beyond the lifetime of a single test instance. Although some of that information has a test program global scope, in many cases there's a strong association between the data and a specific test method. Examples include:

- **Caching expensive (local) data:** In IG-XL, test methods are called from the flow with the test instance properties being handed as arguments. Historically, test methods didn't encourage using state, they were rather considered static, stateless methods. IG-XL is optimized for efficient data handling from its internal caches to the test method (**NOT** repetitively reading from the sheet at run time!).

However, the supported data types are limited to basic ones (`string`, `int`, `bool` and `double`). More complex information (arrays, data structures) must either be encoded in strings or referenced via a key, so that it can be looked up elsewhere. Input data that requires further processing (like generic test methods having to resolve input pin lists and query the instrument types) can't easily cache the results, since it may be called multiple times in the same flow with different arguments.

Repeated re-creation or re-evaluation directly impacts production throughput, unless users find a way to skip that in subsequent runs.

- **Reuse data later in the flow:** Some test concepts involve the (expensive) acquisition of device specific data once, and reuse that multiple times further along on the flow. Such data needs to be reliably re-initialized at the test program start to avoid information leakage to the next DUT. Calibration data (focus, ground) or trimming information fall into this category.
- **Aggregate data across devices:** Certain test techniques involve deliberate information leakage across devices. For instance could a part-average-testing approach dynamically adjust its limits based on the typical behavior of previous devices, so that process outliers are found sooner. Test time optimizations for trim tests could consider the results of the previous DUTs to start with the most-likely result in order to minimize the steps needed.

In all these cases, flexible data storage is required, with awareness of IG-XL events (validation, program start & end, ...) for a robust and reliable solution.

Concept

C#RA adheres to the principle of modularity and encapsulation. Data is stored with the minimum necessary scope and accessibility, preferably bundled into classes providing specific properties and methods to interact with it.

Test Methods are the main place to hold and manage test related data. Test blocks (stateless) receive this data as arguments, and return any results back into the test method for further processing.

Data local to the test method or test class that needs to survive beyond the test instance execution is captured in class fields, with the test class marked with the `Creation.Once` or `Creation.TestInstance` attribute. These make sure that IG-XL re-uses objects instead of creating new, resulting in persistence for any class fields. Access to such data from other test methods is made through an IG-XL API that returns a reference to an instance's test class object.

Key based data storage and retrieval is provided through a dedicated **PersistentStorageService**. Even though this is only needed in certain scenarios, an efficient & robust design is offered to provide a reliable, ready-to-use solution, saving users the effort of implementing their own (see chapter [Persistent Storage Service below](#))

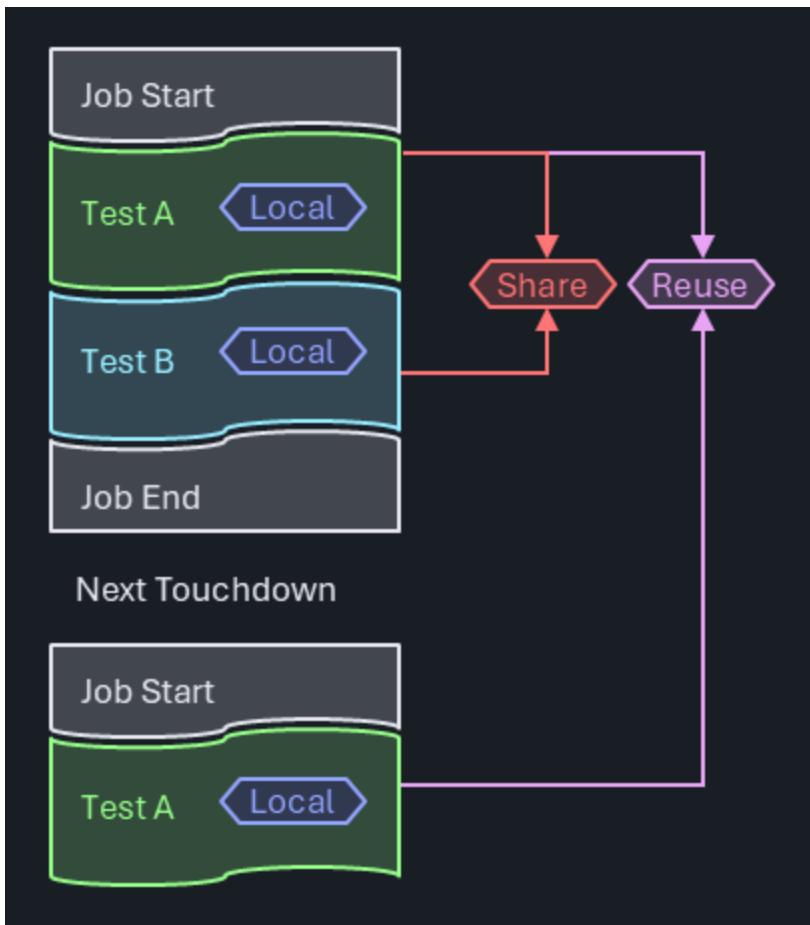
IMPORTANT

C#RA will have persistent storage that users can take advantage of, but it will not store anything in persistent storage by default, users have to opt in and should review the downsides to this storage and minimize it's use to necessities.

Background

VBA

In VBA, data persistence is typically achieved via module-level variables, which are the equivalent of static class fields in .NET. With scope options limited to `private` and `public` (= "global"), basic access control can be implemented. The concept of local static variables in methods may be useful, but can result in poor reusability and maintainability.



There's no limitation in types and no resulting execution time overhead for persistence. The lack of namespaces and effective encapsulation concepts results in convoluted code that violates modularity concepts and bypasses effective data & access protection.

In larger teams and programs, or in code reuse scenarios, these limitations repeatedly cause friction and scalability issues.

.NET

The .NET languages are designed to manage large software solutions, and bring strong concepts like namespaces and the focus on object-oriented principles. In conjunction with IG-XL test programs, a few additional aspects need to be considered:

- Unlike in stand-alone tools (like .NET console apps), the concept of "running" code isn't bifold: .NET assemblies are loaded into the Excel process and "run" together with that, even before a job is started. For interactive scenarios, the debugger is "running", and may keep that running state even after the test program has completed. The question of when objects and data containers are created, preserved, reset or destructed needs consideration.
- Inter-process-communication is required in debug scenarios to synchronize data between the Excel and Debug Run Host process.

IMPORTANT

For normal red-button program runs, .NET test code is loaded into the Excel process and executed there. Using the VisualStudio debugger in that scenario is possible, but has limitations: dynamic code edits are not allowed in breakpoints and the Excel UI and any debug displays are non-responsive (frozen), because the debugger halts the entire process.

For a truly interactive debug experience with debug displays, sheets and to support live code edits, jobs need to be started using the blue Debug-Run button. In that mode, IG-XL will execute .NET test code in a separate process (Debug Run Host or DRH). That process may be halted and even killed anytime without impacting the IG-XL runtime. Such increased flexibility however comes at the cost of overhead. The process needs to be launched, and data (all non-local variables) synchronized in both directions. IG-XL automatically takes care of that, but depending on the amount of data, this (debug only) overhead can become significant.

Encapsulation

One of the fundamental concepts of Object Oriented Programming (OOP) is data encapsulation. It's the exact opposite of a large & global system state.

"Encapsulation in C# is the principle of bundling data (fields) and methods that operate on that data into a single unit (class) while restricting direct access to the internal state by using access modifiers like `private` or `protected`. This ensures controlled interaction with an object's state through public methods or properties, promoting data integrity and simplifying code maintenance."

Encapsulating data in small and independent entities (classes) offering dedicated interfaces for interaction has benefits:

- **Data Protection:** fine access control. The class designer decides which parts shall or shall not be user accessible.
- **Reuse:** goes hand-in-hand with inheritance. Multiple objects of the same type can co-exist without the risk of interference.
- **Modularity:** features can be shared easily as they are contained and have fewer (or no) dependencies.
- **Maintenance:** clear ownership and scope (of code) and a single place to modify for code changes.
- **Abstraction:** functionality is available to users without the need for detailed understanding on how the data is stored / handled internally.
- **Testability:** fewer (or no) dependencies to system state, easier to simulate corner cases.

Persistent Test Class Objects

IG-XL itself uses a mix of programming paradigms, be that for historical reasons or for the fact that device test methodologies often follows a sequential, script-like approach. Some aspects in IG-XL are clearly object oriented (multi-site measurements, capture waveforms, ...), where others follow a more procedural style, like the concept of a Flow executing Test Instances, which call Test Methods. Finally, the use model around [DSPWave](#) expressions applies functional principles, where the output is a consequence of a chain of transforms on inputs without any side effects or external state involved.

.NET introduces the opportunity to better utilize object oriented tools.

In IG-XL, test methods are defined in non-static test classes. To execute a test method, an object has to be created. Static test classes are not supported as they would conflict with the concept of inheritance, which IG-XL TestMethods rely on. From IG-XL 11.00 on, users have a choice of three options for when (and how often) test class objects are created.

Creation.Always - default and only option before IG-XL 11.00

IG-XL will **create a new test class object each time** prior to calling it from the flow, objects are never stored or reused.

This model mimics the VBA approach, but does not allow for object data persistence, because once the test method has completed, the test class object (along with any local variables & class level fields) is disposed off by the garbage collector (GC). Data requiring persistence must be stored elsewhere (breaks encapsulation).

Creation.Once

IG-XL will **create a single object (aka "Singleton") of the test class** the first time it is accessed (Validation, First Run, ...) and reuse it whenever any test instance which uses the class is executed.

Data persistence is available at the class level, but data is common to all test instances using any of the test methods in the class, and shares the lifetime of the Excel process.

This option is well-suited for a collection of related test methods which benefit from a shared data set.

Creation.TestInstance

Separate objects of the test class are created for every test instance the first time each test instance is accessed (Validation, First Run, ...). Those objects are reused whenever their associated test instance is executed.

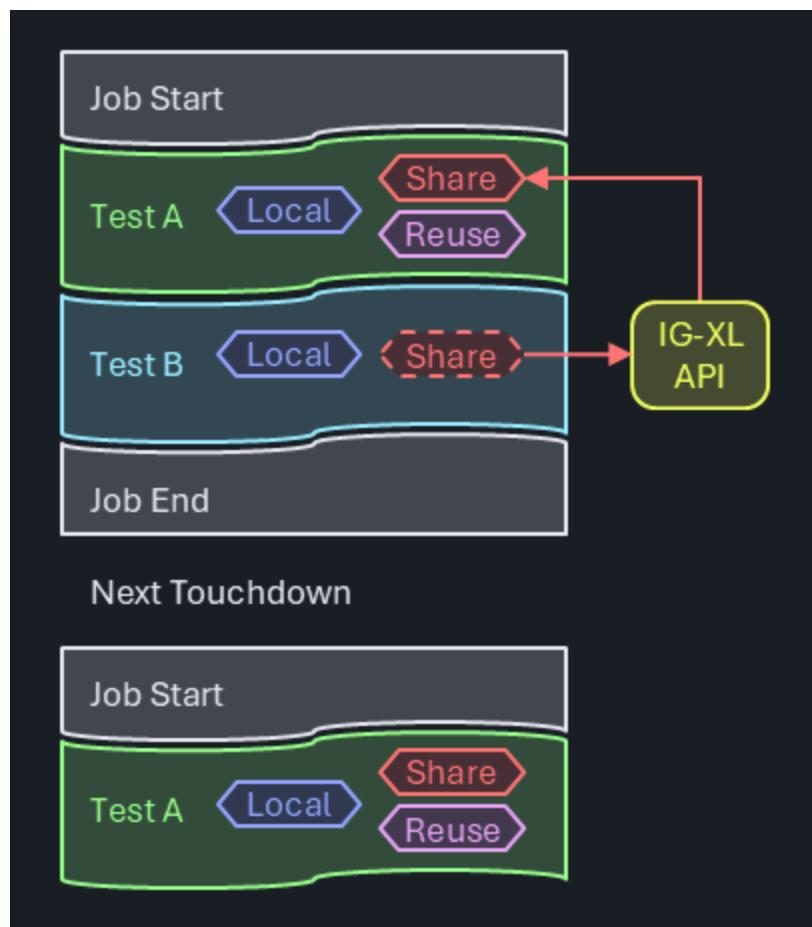
Non-static class level fields allow for fully encapsulated and independent data persistence for each test instance. Different test instance arguments can be pre-processed during validation, and the cached results are locally available for performant production runs. Data doesn't have to be sent and retrieved from somewhere, it's stored right at the place where it's created and processed. Class fields may be used

for object (test instance) specific information, whereas static class fields allow sharing data across all objects.

This model works best for reusable test methods that benefit from out-sourcing some overhead or initialization to first run or validation.

Access from Outside

The author/designer of a class retains fine control over external access. Data intended only for internal use would be defined as `private` fields and would allow no external access. Limited data access, like to only allow writing in certain ranges, or to perform calculations before reading back may go through `public` properties. Unconstrained access is possible when `public` fields are exposed directly (generally not recommended.)



Because IG-XL manages test class objects (initial creation, re-creation on assembly updates), it's important to go through an IG-XL API when accessing such objects from the outside, like from other test methods:

```
Site<int> codeVbg =  
((Trim)TheExec.TestProgram.TestCode.TestClassObjects.GetTestClassObject("TrimVbg", 0)).CodeVbg;
```

NOTE

The exact API language for this is still under review.

Attempts to manually cache this and avoid the IG-XL API is risky, and may result in hard-to-find errors when IG-XL re-created those objects and the cache still holds a reference to a stale object.

IMPORTANT

This API does not open a back-door into IG-XL internals that shouldn't be accessed by users. Instead, it allows the fully OOP compliant use of test class objects. Because IG-XL creates those objects, the user doesn't have access from outside, only from inside.

Not offering this interface would invite users to create their own storage, something like:

```
public static Dictionary<string, TestCodeBase> TestCodeObjects = new();
```

From inside a test method, users could place a handle to itself, and manage the accessors from outside that way:

```
TestCodeObjects.Add(TheExec.DataManager.InstanceName, this);
```

This is risky as IG-XL may re-creates their test class objects without being noticed by the cache, which would then still point to the old (stale) objects. Accessing those will result in hard-to-find errors. Besides, users would redundantly store information that already exists in IG.XL. Better offer read-access to the true source and avoid compromising test solution robustness.

Examples

The first example shows how `Creation.TestInstance` is used to cache an object of the `Pins` class for a simple continuity test:

```
[TestClass(Creation.TestInstance)]
public class Continuity : TestCodeBase {

    private Pins _contPins;

    [TestMethod]
    public void Simple(string pinList, double current, double voltageRange, double waitTime,
        string dibConfig) {
        _contPins ??= new Pins(pinList);
```

```

        Setup.ApplyLevelsTiming();
        Setup.ApplyConfig(dibConfig);
        Setup.Dc.Connect(_contPins);
        Setup.Dc.ForceI(_contPins, current, Measure.Voltage, voltageRange, current);
        Execute.ChrisSuperWait(waitTime);
        PinSite<double> meas = Acquire.Dc.ReadMeter(_contPins);
        Setup.Dc.Disconnect(_contPins);
        Datalog.TestParametric(meas, "V");
    }
}

```

The `new Pins(pinList)` constructor is only executed when the `_contPins` object is `null`, and then stored at the class level for this test instance. It goes through relatively expensive code that resolves the pin list and identifies the underlying instrument types behind every single pin. That object is used in the subsequent, instrument agnostic methods (test blocks) to perform the desired actions.

The second example presents a trim test utilizing the `Creation.Once` feature. The test method `VoltageBandgap()` sets up the device, performs a linear sweep through all trim codes and uses SiteGenerics expressions to determine the best trim code - which is stored in the class level field `_codeVbg`:

```

[TestClass(Creation.Once)]
public class Trim : TestCodeBase {

    private const int _defaultVbg = 7; // center value, until we know better
    private Site<int> _codeVbg = new(_defaultVbg);

    public Site<int> CodeVbg => _codeVbg;

    [ExecInterpose_OnProgramStarted]
    public void InitTrimValues() => _codeVbg.Fill(_defaultVbg);

    [TestMethod]
    public void VoltageBandgap(double targetV) {
        // connect resources
        // perform sweep
        Site<Samples<double>> cap = null; //readback all acquired strobes
        _codeVbg = cap.Select(s => {
            TerMath.Abs(s - targetV).Min(out int index);
            return index;
        });
        Datalog.TestParametric(_codeVbg);
        Datalog.TestParametric(cap.Select((m, site) => m[_codeVbg[site]]), "mV");
    }
}

```

```
    }
}
```

That field is defined as `private` so it can not be directly accessed from the outside. It can however be read via the read-only `public Site<int> CodeVbg => _codeVbg;` property. Additional logic is implemented, so that the trim code is re-initialized to its default value on every program start (method `InitTrimValues()` decorated with attribute `[ExecInterpose_OnProgramStarted]`).

Now consider a general power-up function, which needs to write the "currently best known trim code" into the DUT. When called from the flow **before** trimming was executed, the default trim value shall be used. After trimming, the actual trim code (per site) is applied. Regardless of when this method is called, it'll always access the true data source:

```
[TestMethod]
public void PowerUp() {
    Setup.ApplyLevelsTiming();
    var vbgTrimObject = TheProgram.TestCode().TestClassObjects.GetObject<Trim>
(trimVbgTestInstance);
    Site<int> code = vbgTrimObject.CodeVbg;
    Execute.Digital.WriteRegister(registerVbg, code);
}
```

.NET inherent access control as the class author defined it is maintained. Type mismatches or attempts to bypass is flagged by the compiler:

```
// compiler will flag type mismatches
Site<double> codeWrongType = vbgTrimObject.CodeVbg;

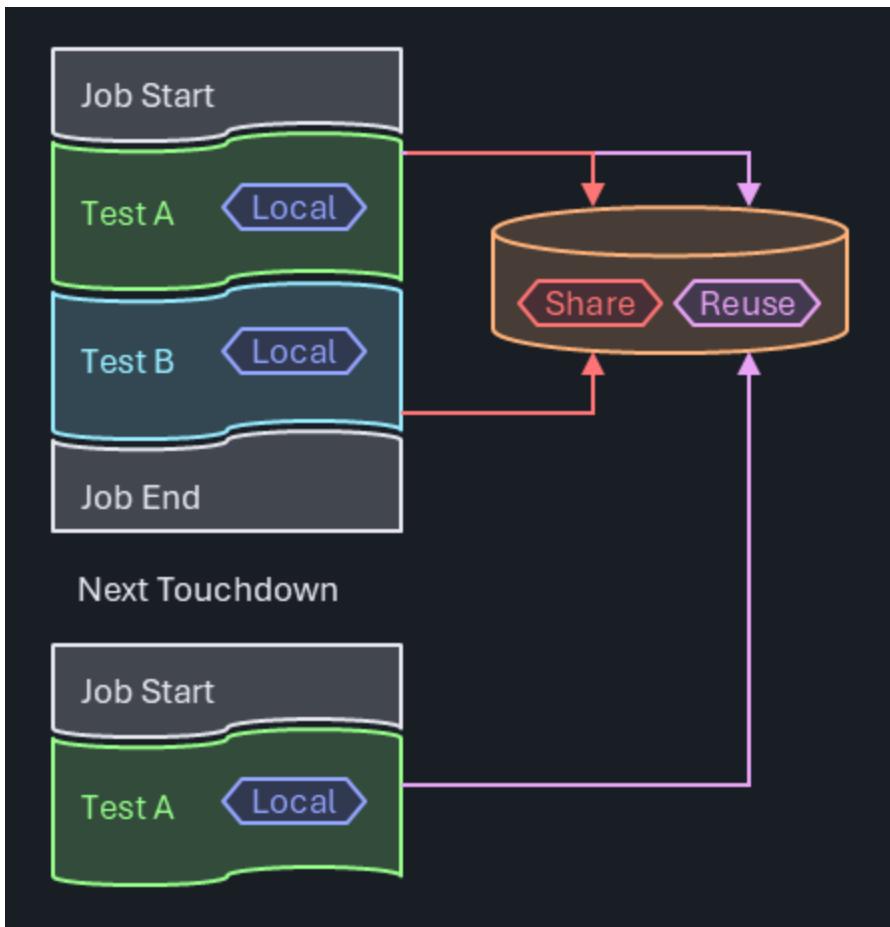
// overwriting / deleting read-only not allowed
vbgTrimObject.CodeVbg = new Site<int>(5);

// access to private backing field not allowed
vbgTrimObject._codeVbg = 5;
```

Persistent Storage Service

Problem Statement

In this project, Object-Oriented Programming (OOP) is primarily used. However, there are corner cases where it is necessary to store generic information that does not belong to any specific object. To address this, a singleton data storage solution has been implemented using a `Dictionary<string, object>` with methods for unboxing to maintain type safety.



Use Model

- **Initialization:** The singleton instance is created and initialized at first use.
- **Data Storage:** The `Dictionary<string, object>` is used to store data. Methods are provided for adding, retrieving, and unboxing data to specific types.
- **Access:** The singleton instance can be accessed from anywhere in the application to store or retrieve data.

Open Questions

Even with great care, the following downsides still remain - and have to be accepted when used:

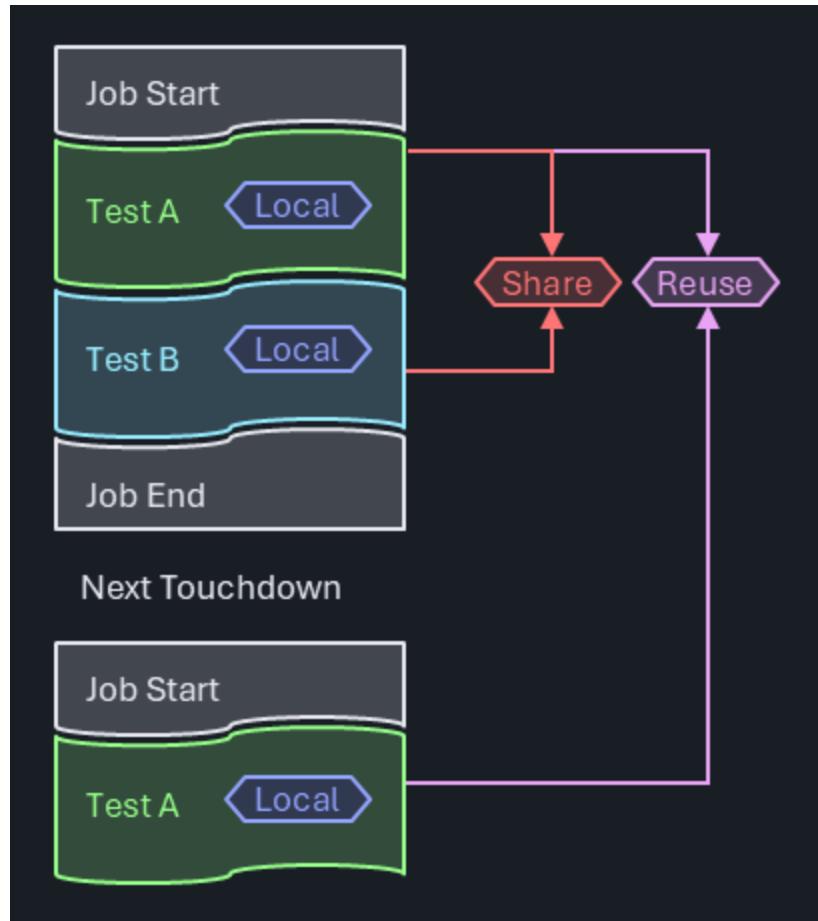
- **Performance Overhead:** Boxing and unboxing add some performance overhead.
- **Modularity:** The use of a singleton can reduce modularity and make unit testing more challenging.
- **Scalability:** How will the data storage scale with increasing data and concurrent access?
- **Integration:** Need to create interfaces and manage integration into IG-XL Execip events.
- **Debuggability:** An additional level of indirection impacts debuggability, necessitating a debugger viewer for the contents.

Alternatives considered

The following alternative was analyzed. In the overall assessment of the pros and cons they were considered less attractive than the proposed solutions for the reasons documented.

Use of Global .NET Variables

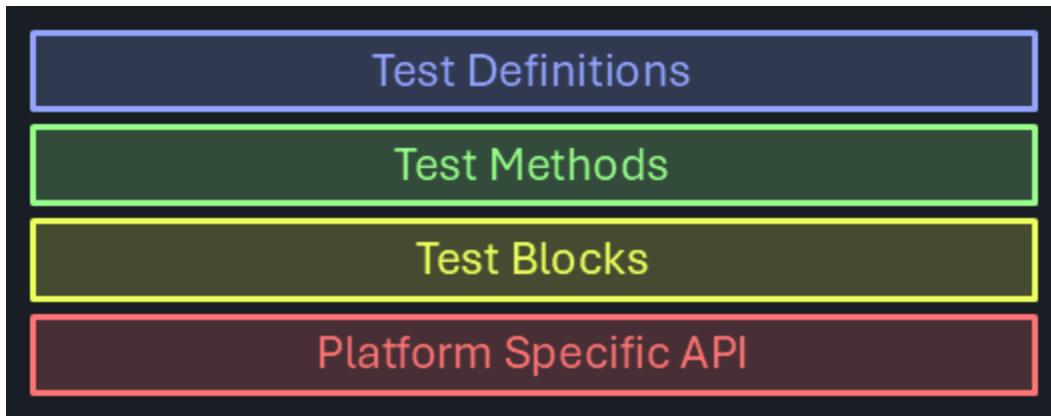
Being a little awkwardly named, **Global .NET Variables** do add persistence to objects in blue button runs involving the DRH process. Specifically, all serializable `public static` and `private static` objects are synchronized between the Excel and DRH process, so that the fundamental requirement of persistence is fulfilled.



The limitation to `static` however collides with the requirement of storing data **per test instance**. The use case of caching local data from validation or first run in a test instance would not (easily) be possible. The model is good for data that needs to be shared across a test class, providing zero overhead for read and write access.

Platform Independence

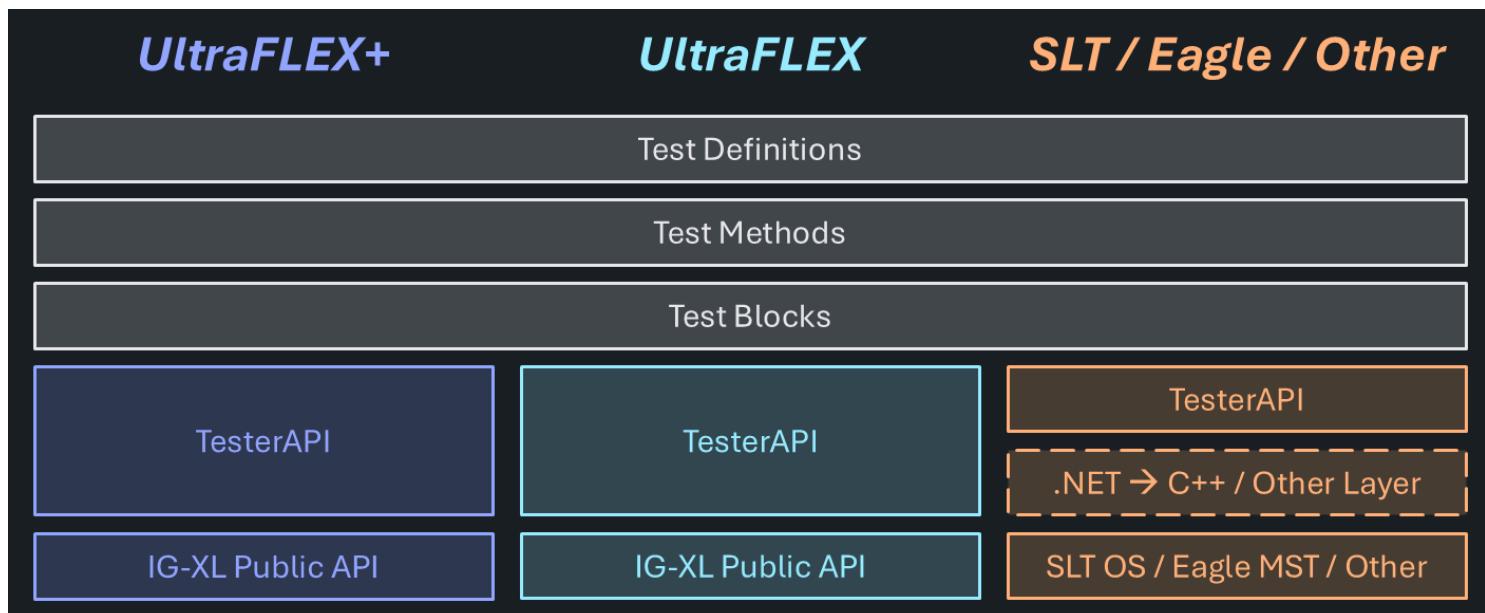
To achieve platform independence, meaning identical test coverage and results on multiple, possibly very incompatible testers, it's required to introduce commonality **somewhere**. And there are multiple places in a typical [test abstraction stack-up](#) where that could be done.



Hardware Abstraction Layer

"A hardware abstraction layer (HAL) is a software layer that provides a consistent interface between hardware components and higher-level software, isolating the operating system or applications from the specifics of the hardware."

TesterAPI (FlexTest) creates an interface by abstracting the Public API on IG-XL based testers. On other testers, it is intended to achieve the same, but an additional language / runtime translation layer may be required to interface to C++ / Java based or other platforms.



Since TesterAPI offers a common interface, the layers above can be common for all platforms - the same test program can run everywhere.

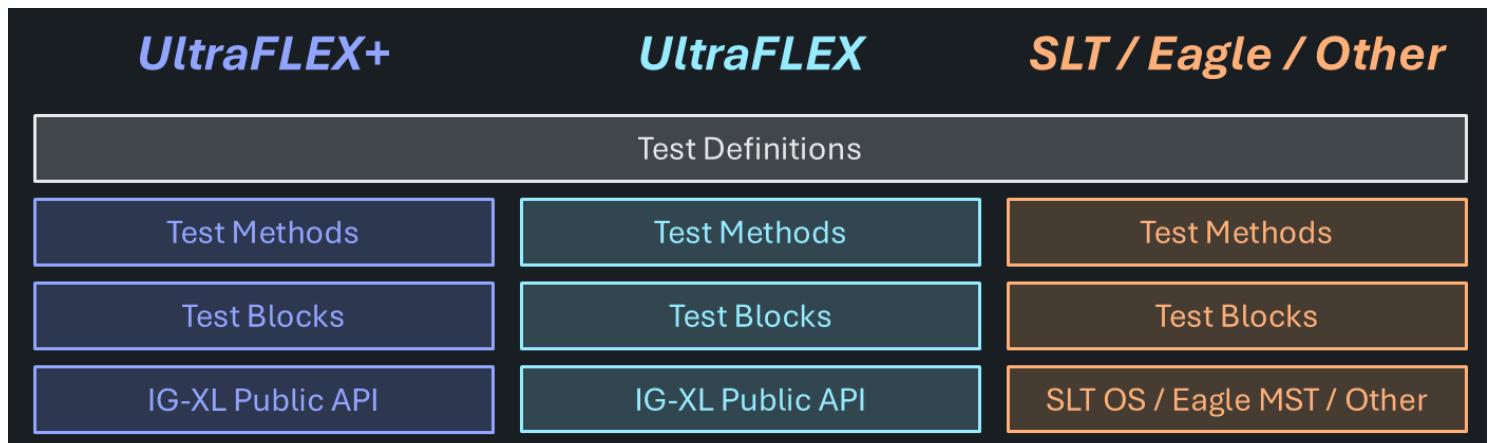
Uniform Test Methods

Enforcing a common and exclusive interface to the tester hardware poses some challenges - which may be hard to overcome:

- an identical user interface for all testers is difficult for features that only exist on some, and constrains the feature set to the common denominator. Offering any specific functionality on the top level would violate platform agnosticism, as code that uses them is now not anymore generic. Underneath though, specific features can be accessed, if they can be contained and have no implications on the user level (like parameters that need to be set).
- every additional layer introduces overhead and a level of indirection. For users who focus on a single platform only, that complexity could be a negative factor, without any apparent benefit.

Overall, test programs following that approach will have a competitive disadvantage over specifically implemented ones using the hardware and software features directly.

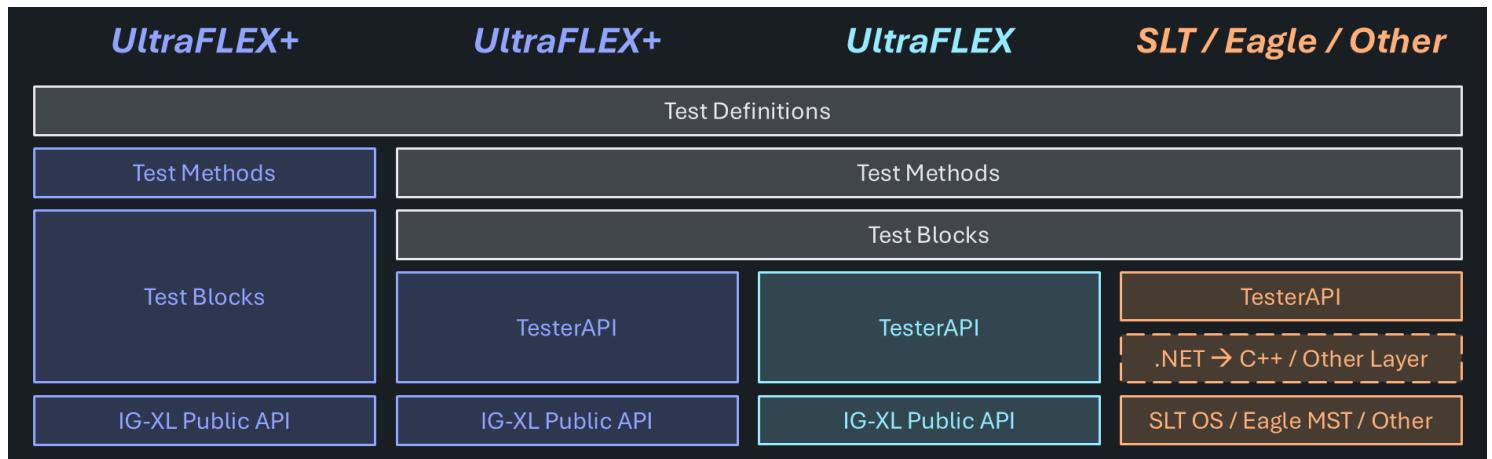
However, platform independence is still present at the test definition layer in this model. Platform specific implementations for the Test Method and Test Block levels may offer a common interface, but utilize hardware & software features natively.



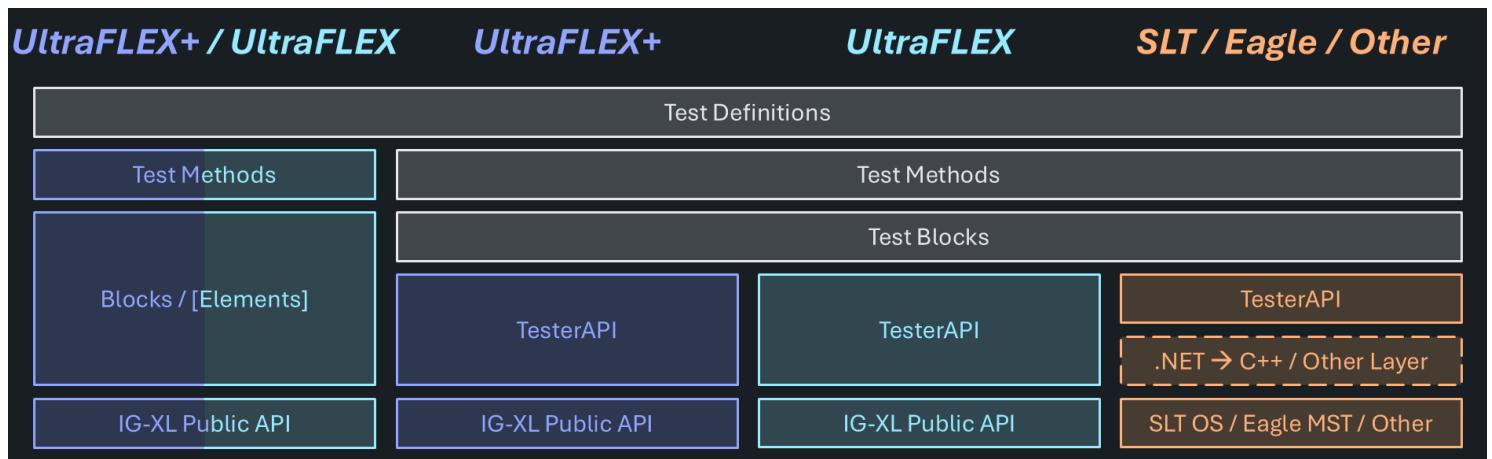
In that scenario it's not the same test program that runs on any platform, but a test program covering the same functionality can be generated. Since they are directly comparable, the two approaches can be used to compare results and benchmark performance.

Combining the Benefits

As it's often the case, sweet spots may exist between the extreme positions. At the cost of some redundancy, both paths can be combined, offering more flexibility.



Some testers (like UltraFlex and UltraFlex+) which share a majority of the platform specific API, it may be possible to create a common, specific implementation that handles (little) platform specifics inside. It is expected that this can be achieved without compromising performance and complexity.



As additional benefit, there would now be a way to compare and benchmark the platform independent vs. specific implementation under realistic scenarios. That would take much of the guesswork out of test time critical projects, which are not yet where they need to be. Such insight could be helpful to identify specific areas to scrutinize in a TTR activity.

SSN - Streaming Scan Network

For Streaming Scan Network, the **STIL** patterns often comes in a pack of files with 5 or more types:

- Test Setup Pattern
 - used to precondition DUT into scan test mode, setup clock, etc.
 - This specific pattern is usually common to multiple SSN tests and can be executed separately
- SSN Setup Pattern
 - used to config the **SSN BUS** by configuring each node on the bus, nodes include but not limited to :
 - **SSH** : Streaming Scan Host
 - **SMUX**: path branching multiplexer
 - config is typically done through **iJTAG** via **ICL** command - Instrument Connectivity Language
 - Each **SSH** or **SMUX** have their individual ICL interface and can be programmed by IJTAG
 - in case of OCComp, a particular SSH can be turned off by config the **disable_contribution_bit** of that SSH, this is crucial for fail diagnosis.
- SSN Payload Pattern
 - contains only the data on the **SSN BUS**
 - Contains both SCAN IN and SCAN OUT pins data
 - scan data are weaved/scrambled on the entire BUS and packed back to back without clear boundary of shift or capture
 - each SSH on the bus will take data at certain cycles on certain BUS pins and reconstruct the scan load following a certain algorithm, similarly for the scan unload.
 - the algorithm to figure out a certain bit at a certain cycle on a certain pin corresponding to which **EDT** (Embedded Deterministic Test) is available to ATE via a **csv** file.
 - The SSN bus operation can be reconfigured from one SSN test to another by means of the SSN setup pattern. The algorithm is passed to the device by means of **iJTAG** and to the tester via **csv** file, therefore the **csv** files are unique per pattern and might be uniform or different
- SSN End Pattern
 - optionally used to reset the SSH on the active SSN BUS
 - can optionally read the status registers on each SSH
 - this is the case where OCComp test results are fetched through the **iJTAG** port.
- Test End Pattern
 - used to conclude the test mode, restore the DUT, really case dependent.

Flavors

OnChipCompare (OCCOMP)

If the DUT contains multiple identical cores or EDTs, the actual data that goes into each identical EDT is also identical. Therefore, it makes sense to send only one copy of the **stimulus data** and configure all the SSHs within the group of identical EDTs to pick the data from the same slot on the SSN BUS.

Similarly, the **Expected data** and **Mask data** should also be identical. Instead of letting the ATE perform the comparison, the SSH (with OCComp capability) can take the Expected and Mask data and perform the comparison individually for each core. Once a mismatch is observed, the status will be recorded by **SSH** and can be fetched later via **iJTAG**; this particular mismatch status is called a **Sticky Bit**. The Pass/Fail status for each core and each scan compared data (scan cell) is or-ed among all the identical EDT of the same group if they contribute. The cumulative Pass/Fail status is then propagated on the SSN BUS and observed by the ATE on the SSN bus OUT pins Only part of the data on SSN bus OUT pins are actual cumulative Pass/Fail status.

The mapping information for the locations of the **contribution_bit** or **sticky_bit** is also available in the form of a **CSV** file, which is generated alongside the **ATP** file from the **STIL** file. Currently, obtaining the failed core information is a manual process. By using a code library provided on EK (originally authored by Chris Cassidy), users can parse the CSV file and determine the failed cores by providing the failed cycle numbers on the JTAG TDO pin after bursting the ssn_end pattern.

Example of a STIL file section that is used for generating the OCComp meta data(*_ssn.csv)

► STIL File Example

If ALL the SSH are off then no OCComp meta data(*_ssn.csv) is created

```
Ann {*} on_chip_compare = off {*} 
```

TesterCompare (TC)

For SSN payload STIL patterns, it is almost certain that a CSV file will be generated alongside the ATP file. This CSV file contains the algorithm that describes how the pin and cycle data are rotated and mapped to each core (in IG-XL 11.0; in the future, the mapping will need to be at the SSH level rather than the core level).

In the case of grouped identical core OCComp test patterns, there is no way to identify which SSH instance in the group is causing a bit on the scan-out bus to flip. As a result, the per-core pass/fail result cannot be derived from the scan-out fail pin or cycles. In the STIL file, within the **Active_Ssh_Section**, there will be a **representative_ssh** that indicates which SSH/ICL instance is representing the group on the SSN bus if that SSH/ICL instance is sharing the pin/cycle slots with other SSH/ICL instances. In this case, only the representative SSH will be listed in the TC section **Ann {*} TESSENT_PRAGMA ssn_mapping - begin/end {*}** with the pin/cycle rotation information. This is useful for diagnosis because the user needs to disable all but one SSH that shares the same representative SSH.

However, for unique cores, it is more efficient to let the tester perform the comparison. In this case, IG-XL will parse the CSV file, capture the fail bits (pin/cycle), and generate the failed core information natively. The API to retrieve the result is:

```
var ssnResults = TheHdw.Digital.Patgen.ReadScanNetworkResults(); 
```

Example of stil file section that generate the TesterCompare meta data(*_ssn_mapfile.csv)

► STIL File Example

Combo (TC + OCCComp)

From a device DFT perspective, it does not make sense to generate tests that are either OCCComp only or TC only. It is possible that the device does not have any identical cores at all, in which case all patterns would be TC only. However, it is very unlikely that a device would have only identical cores, so in practice, only OCCComp test patterns would rarely be provided.

This means that in a real customer project, SSN patterns are either:

- TC only, only the TesterCompare meta data(*_ssn_mapfile.csv) is created
- TC + OCCComp combo, in that case 2 csv files will be created, the TesterCompare meta data(_ssn_mapfile.csv) and the OCCComp (_ssn.csv)

IG-XL can always parse the TC portion natively, but currently handling the OCCComp we have to use the code lib discussed above. This is the main reason we need a unified solution to cover both case in **CSRA**.

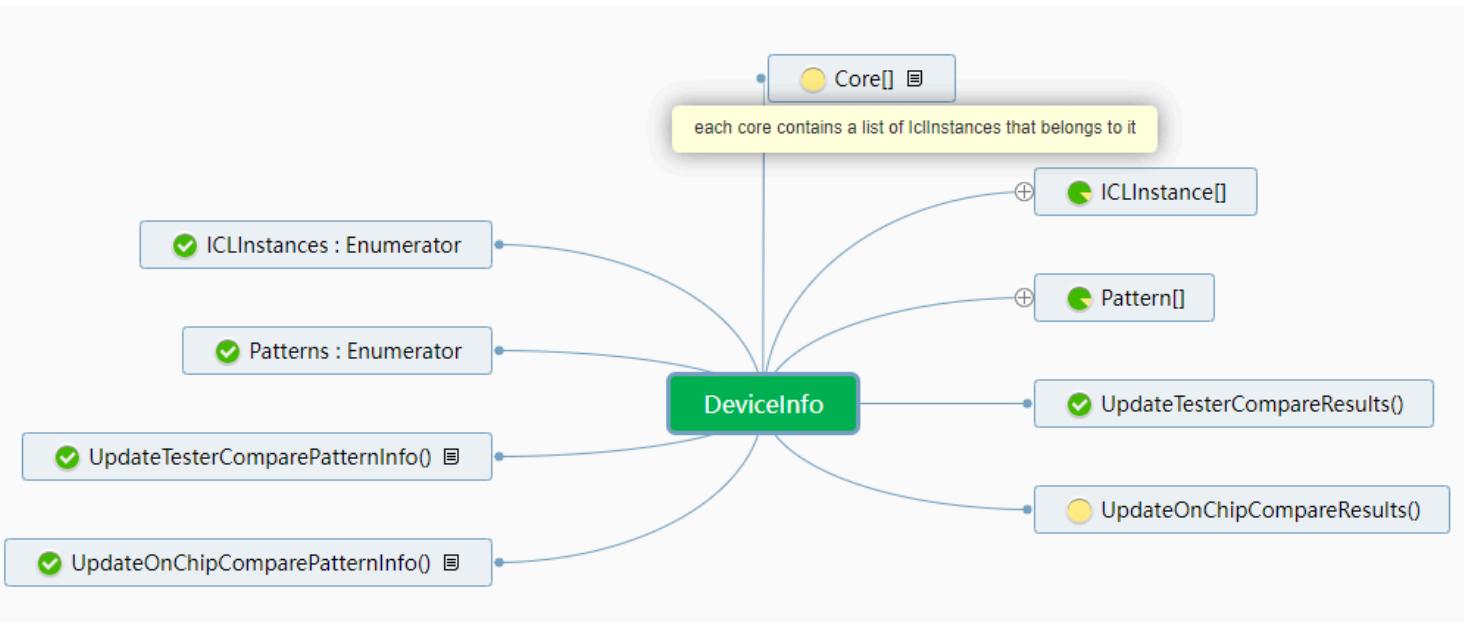
Implementation

Implementation steps

- Drop 1
 - implement initial support for OCCComp and TC
 - OCCComp CSV support only
- Drop 2
 - implement initial support for OCCComp/TC combo
 - OCCComp pattern Tags support
 - implement DeviceInfo class
- Drop 3
 - implement **per-chain sticky bit** requirements
 - TBD

One super class

Similar to TheHdw, which is the portal to all tester instruments, and TheExec, which is the interface to all software controls, we propose to use one super object to handle all DUT related info so that test results can have a traceable history. At a high level, we propose something like **DeviceInfo** as shown in below:



Essentially it is a sparse matrix that can be indexed by core/ssh and pattern:

The Map (for visual illustration)		Pattern List					
Core Instance	iclInstance	intermediate_level	mixed_identical_nonidr	identical_top	Pat_4	Pat_5	Pat_6...
SubSystemA	subsystemA.subsystem_rtl_tessent_ssn_scan_host_1_inst	i1					
SubSystemA	subsystemA.corec_1.unique_core_rtl_tessent_ssn_scan_host_1_inst	i2					
SubSystemA	subsystemA.corec_1.identical_core_rtl_tessent_ssn_scan_host_1_inst	i3					
SubSystemA	subsystemA.corec_2.unique_core_rtl_tessent_ssn_scan_host_1_inst	i4					
SubSystemB	subsystemB.subsystem_rtl_tessent_ssn_scan_host_1_inst	i5					
SubSystemB	subsystemB.coreb_1.unique_core_rtl_tessent_ssn_scan_host_1_inst	i6					
SubSystemB	subsystemB.corec_1.unique_core_rtl_tessent_ssn_scan_host_1_inst	i7					
SubSystemB	subsystemB.corec_2.identical_core_rtl_tessent_ssn_scan_host_1_inst	i8					
subsystemA/coreb_i1	subsystemA.coreb_i1.unique_core_rtl_tessent_ssn_scan_host_1_inst		i9	i15			
subsystemA/corec_i1	subsystemA.corec_1.unique_core_rtl_tessent_ssn_scan_host_1_inst		i10	i16			
subsystemA/corec_i2	subsystemA.corec_2.unique_core_rtl_tessent_ssn_scan_host_1_inst		i11	i17			
subsystemB/coreb_i1	subsystemB.coreb_i1.unique_core_rtl_tessent_ssn_scan_host_1_inst		i12	i18			
subsystemB/corec_i1	subsystemB.corec_1.identical_core_rtl_tessent_ssn_scan_host_1_inst		i13	i19			
subsystemB/corec_i2	subsystemB.corec_2.identical_core_rtl_tessent_ssn_scan_host_1_inst		i14	i20			

The thought is that, during flow run we will need to update the elements by pattern index, and at the end we will need to gather pass/fail result by cores, where each **core** is essentially a list of icl/ssh instances.

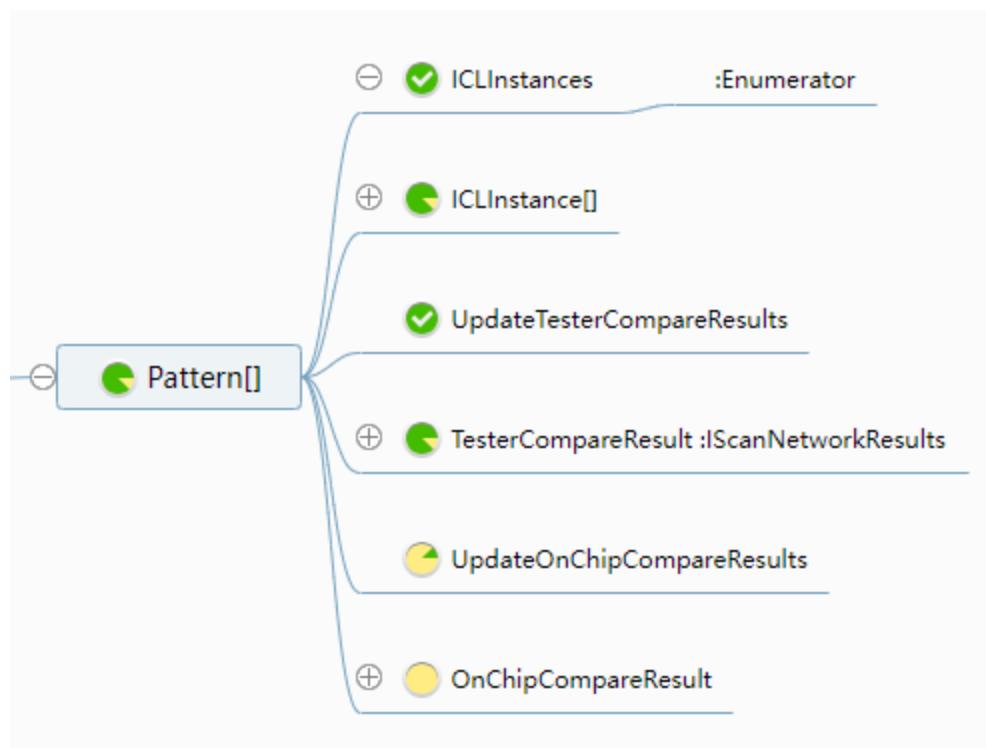
```

// ssh/icl instance can be indexed by pattern or by core/icl instance
//DeviceInfo.Pattern["identical"].IclInstance["coreb_i1"]...; // during test, index by
pattern first then by icl
//DeviceInfo.IclInstance["coreb_i1"].Pattern["identical"]...; // during binning, index
by icl/core then by pattern
//DeviceInfo.Pattern["identical"].IclInstance["coreb_i1"] and
DeviceInfo.IclInstance["coreb_i1"].Pattern["identical"] are pointing to the same object.

// Access per core test result example:
foreach(var ssh in DeviceInfo.Core["SubsystemA"]) {
    //DeviceInfo.IclInstance[ssh].Results...
}
  
```

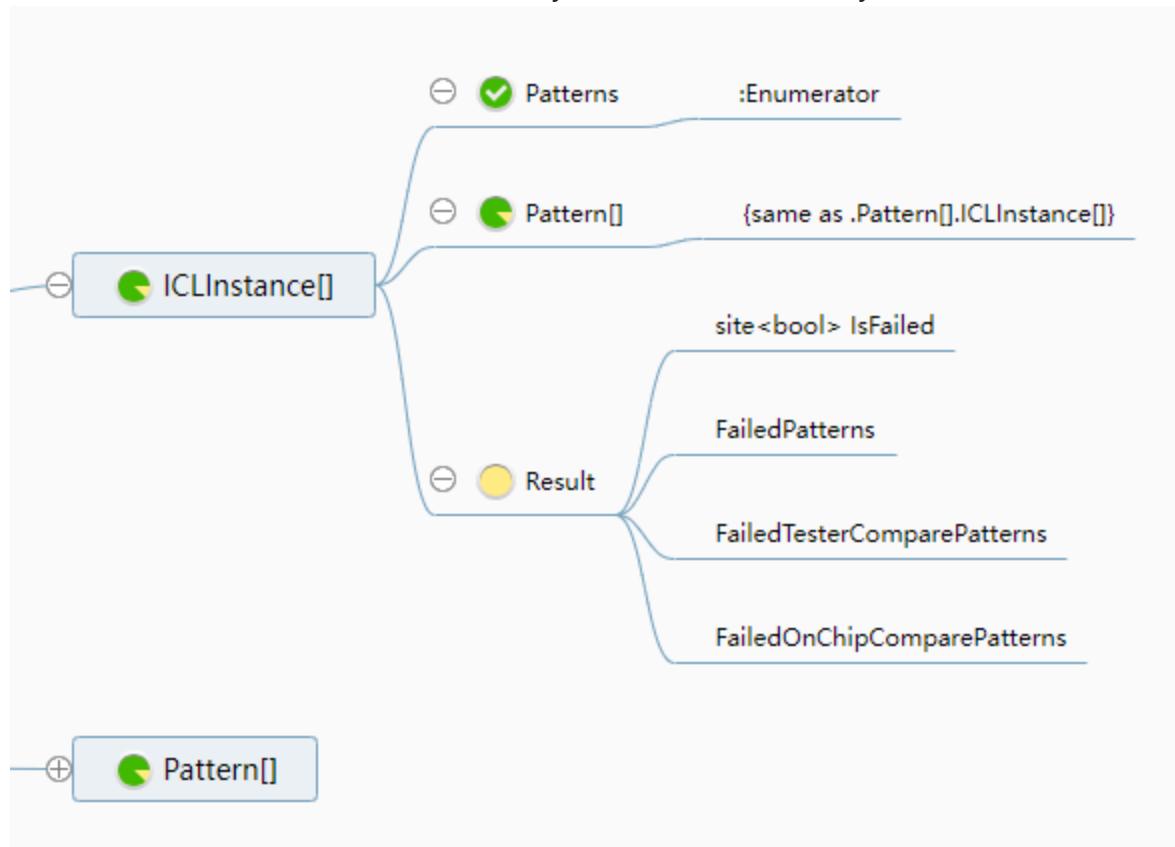
- Currently(IG-XL 11.0), after each ssn pattern set burst, instead of just one pass/fail patgen result, there will be per core/ssh pass fail results returned by Tester Compare and by On-Chip Compare. Thus we need something like:

```
DeviceInfo.Pattern[PatSet].UpdateTesterCompareResults(); // for Tester Compare  
DeviceInfo.Pattern[PatSet].UpdateOnChipCompareResults(); // for On-Chip Compare
```

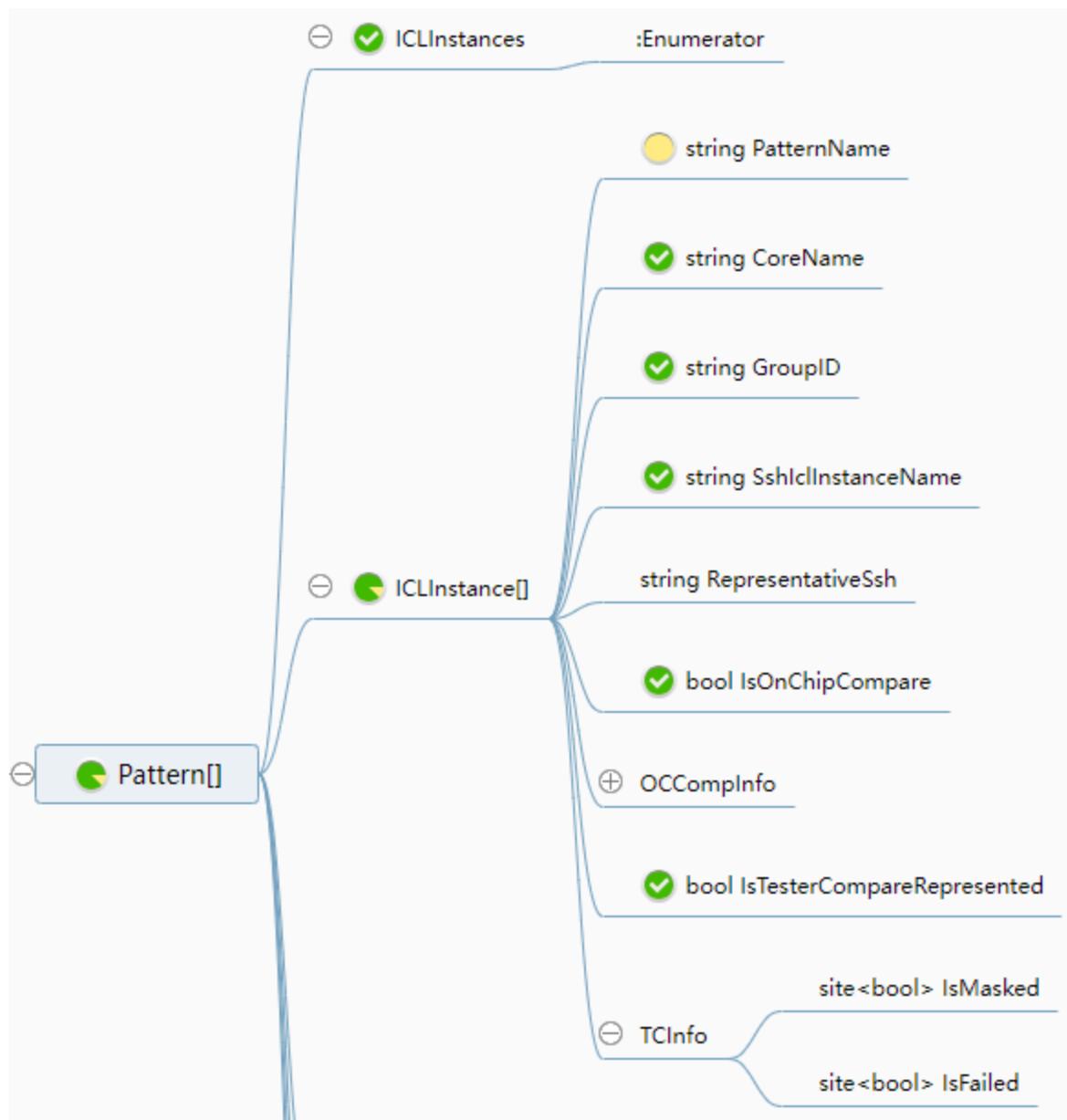


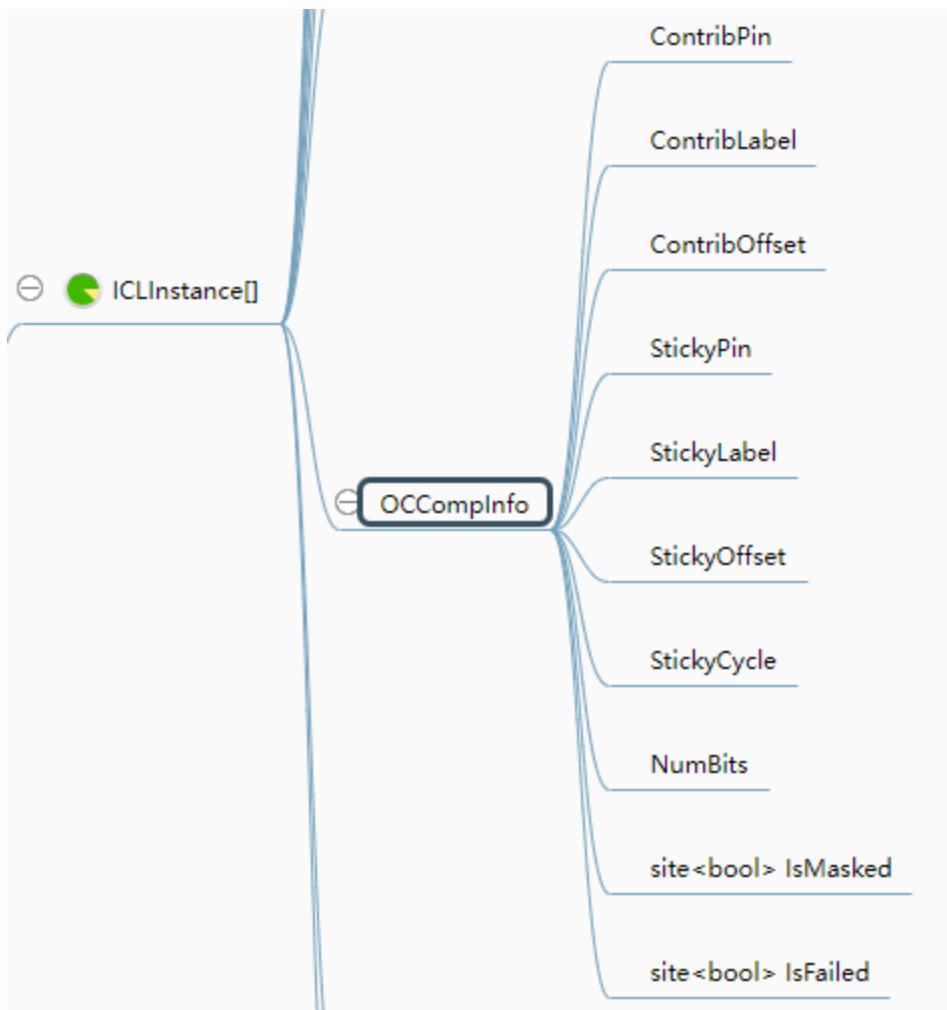
```
// after pattern burst, read the TesterCompare ssn results:  
var ssnTcResults = TheHdw.Digital.Patgen.ReadScanNetworkResults();  
DeviceInfo.UpdateTesterCompareResults(ssnPatternSet, ssnTcResults);  
// alternatively:  
DeviceInfo.Pattern[ssnPatternSet.Value].UpdateTesterCompareResults(ssnTcResults);  
// code for updating OCCOMP test result is TBD
```

- Elements shall be able to be indexed by icl/ssh instance firstly as well:



- Finally, the per pattern per ssh element will have the detailed info such as contribution/sticky bits and pass/fail/masked flags.





- The full list of properties is [TBD]

Property Name	type	Meaning
PatternName	string	name of pattern/set for this element
CoreName	string	name of the core which this icl/ssh belongs to in this pattern
SshIclInstanceName	string	name of the ssh/icl instance, should use ssh? TBD
representative ssh	string	name of the representative ssh for this ssh, blank if this ssh is on TC
OCCComp enabled	bool	true if <code>on_chip_compare = on</code> (in stil)
ContributionPin	string	empty string if <code>on_chip_compare = off</code>
ContributionLabel	string	empty string if <code>on_chip_compare = off</code>
ContributionOffset	string	empty string if <code>on_chip_compare = off</code>
StickyPin	string	empty string if <code>on_chip_compare = off</code>

Property Name	type	Meaning
StickyLabel	string	empty string if <code>on_chip_compare = off</code>
StickyOffset	string	empty string if <code>on_chip_compare = off</code>
StickyCycle	string	empty string if <code>on_chip_compare = off</code>
OCComp Masked	<code>site<bool></code>	= <code>true</code> if <code>disable_contribution_bit</code> is asserted
OCComp Failed	<code>site<bool></code>	= <code>true</code> if <code>sticky_bit</code> failed
TC Represented	bool	= <code>true</code> if this ssh/icl has a representative ssh
TC Masked	<code>site<bool></code>	= <code>true</code> if ssh/icl is masked by IG-XL (mask is at core level)
TC Failed	<code>site<bool></code>	= true if <code>ReadScanNetwork()</code> report failure (fail is at core level)
fail_cycle		TBD
fail_pin		TBD

- Example:

ict/pat ID	pattern name	Core instance	ictd instance name	Representative ssh	OCCOMP enabled	Control pin	Control offset	Control label	Sticky pin	Sticky offset	Sticky cycle	Sticky init	OCCOMP masked	OCCOMP result	TC represented	TC masked	TC result	Num bits
1	intermediate	subSystemA	subSystemA	subSystemA.core_1.unique.core_rt.tsl(subSystemB.core_1.unique)	Y	[tag.tdi]	63	enable_combination0	[tag.tdo]	101011	sticky_status0	Y	1	1	1	1	1	1
2	intermediate	subSystemA	subSystemA	subSystemA.core_1.unique.core_rt.tsl(subSystemB.core_1.unique)	Y	[tag.tdi]	851	enable_combination0	[tag.tdo]	24	107985_sticky_status0	Y	1	1	1	1	1	1
3	intermediate	subSystemA	subSystemA	subSystemA.core_1.unique.core_rt.tsl(subSystemB.core_1.unique)	Y	[tag.tdi]	666	enable_combination0	[tag.tdo]	16	107977_sticky_status0	Y	1	1	1	1	1	1
4	intermediate	subSystemB	subSystemB	subSystemB.core_1.unique.core_rt.tsl(subSystemA.core_1.unique)	Y	[tag.tdi]	491	enable_combination0	[tag.tdo]	8	107983_sticky_status0	Y	1	1	1	1	1	1
5	intermediate	subSystemB	subSystemB	subSystemB.core_1.unique.core_rt.tsl(subSystemA.core_1.unique)	Y	[tag.tdi]	291	enable_combination0	[tag.tdo]	50	107983_sticky_status0	Y	1	1	1	1	1	1
6	intermediate	subSystemB	subSystemB	subSystemB.core_1.unique.core_rt.tlessn.ssn.scan.host_1.inst	Y	[tag.tdi]	1412	enable_combination0	[tag.tdo]	54	107913_sticky_status0	Y	1	1	1	1	1	1
7	intermediate	subSystemB	subSystemB	subSystemB.core_1.unique.core_rt.tlessn.ssn.scan.host_1.inst	Y	[tag.tdi]	1227	enable_combination0	[tag.tdo]	46	107907_sticky_status0	Y	1	1	1	1	1	1
8	intermediate	subSystemB	subSystemB	subSystemB.core_1.unique.core_rt.tlessn.ssn.scan.host_1.inst	Y	[tag.tdi]	1020	enable_combination0	[tag.tdo]	20	107907_sticky_status0	Y	1	1	1	1	1	1
9	mixed	subSystemAcore_1B	subSystemAcore_1B	subSystemAcore_1B.unique.core_rt.tlessn.scan.host_1.inst	Y	[tag.tdi]	63	enable_combination0	[tag.tdo]	8	16686_sticky_status0	Y	1	1	1	1	1	1
10	mixed	subSystemAcore_1C	subSystemAcore_1C	subSystemAcore_1C.unique.core_rt.tlessn.scan.host_1.inst	Y	[tag.tdi]	248	enable_combination0	[tag.tdo]	16	16694_sticky_status0	Y	1	1	1	1	1	1
11	mixed	subSystemAcore_1D	subSystemAcore_1D	subSystemAcore_1D.unique.core_rt.tlessn.scan.host_1.inst	Y	[tag.tdi]	433	enable_combination0	[tag.tdo]	24	16702_sticky_status0	Y	1	1	1	1	1	1
12	mixed	subSystemAcore_1E	subSystemAcore_1E	subSystemAcore_1E.unique.core_rt.tlessn.scan.host_1.inst	Y	[tag.tdi]	809	enable_combination0	[tag.tdo]	39	16717_sticky_status0	Y	1	1	1	1	1	1
13	mixed	subSystemBcore_1I	subSystemBcore_1I	subSystemBcore_1I.unique.core_rt.tlessn.scan.host_1.inst	Y	[tag.tdi]	61	enable_combination0	[tag.tdo]	8	17741_sticky_status0	Y	1	1	1	1	1	1
14	mixed	subSystemBcore_1O	subSystemBcore_1O	subSystemBcore_1O.unique.core_rt.tlessn.scan.host_1.inst	Y	[tag.tdi]	248	enable_combination0	[tag.tdo]	35	17753_sticky_status0	Y	1	1	1	1	1	1
15	identical	subSystemAcore_1I	subSystemAcore_1I	subSystemAcore_1I.unique.core_rt.tlessn.scan.host_1.inst	Y	[tag.tdi]	433	enable_combination0	[tag.tdo]	24	17761_sticky_status0	Y	1	1	1	1	1	1
16	identical	subSystemBcore_1I	subSystemBcore_1I	subSystemBcore_1I.unique.core_rt.tlessn.scan.host_1.inst	Y	[tag.tdi]	624	enable_combination0	[tag.tdo]	38	17775_sticky_status0	Y	1	1	1	1	1	1
17	identical	subSystemAcore_1Z	subSystemAcore_1Z	subSystemAcore_1Z.unique.core_rt.tlessn.scan.host_1.inst	Y	[tag.tdi]	994	enable_combination0	[tag.tdo]	45	17782_sticky_status0	Y	1	1	1	1	1	1
18	identical	subSystemBcore_1Z	subSystemBcore_1Z	subSystemBcore_1Z.unique.core_rt.tlessn.scan.host_1.inst	Y	[tag.tdi]	994	enable_combination0	[tag.tdo]	54	17791_sticky_status0	Y	1	1	1	1	1	1

- The object shall be initialized during validation:

```
// during validation, read and process the csv files
// Create an On Chip Compare object tied to the tested pattern/set
TheHdw.Digital.AddOnChipCompare(ssnPatternSet.Value);
var occPatInfo = TheHdw.Digital.OnChipCompare(ssnPatternSet.Value);

// Read the CSV files associated with the Sticky Pattern and the Contribution Pattern
occPatInfo.ReadCSV(stickyCsv);
occPatInfo.ReadCSV(contribCsv);

//Set the Contribution Pattern to track what pattern to modify during re-bursting
TheHdw.Digital.OnChipCompare(ssnPatternSet.Value).SetContributionPattern(contribPatName)
DeviceInfo.UpdateOnChipComparePatternInfo(ssnPatternSet.Value, occPatInfo);

// for Tester Compare:
```

```
var tcCores = TheHdw.Digital.ScanNetworks[ssnPatternSet.Value].CoreNames;
DeviceInfo.UpdateTesterComparePatternInfo(ssnPatternSet.Value, tcCores);
```

- The OCCComp part will be changed to something similar with TC.
- The TC part may be changed after 11.0 due to need for ssh/icl level of details.

• Partial Good Binning

At the end of Test flow, customer would need to know the test result on a per core basis, and bin to different grade base on a rule set by customer.

The Map (for visual illustration)		Pattern List						
Core Instance	icl Instance	intermediate_level	mixed_identical_nonidler	identical_top	Pat_4	Pat_5	Pat_6...	
SubSystemA	subsystemA.coreb_i1.unique_core_rtl_tes	i2						
subsystemA/coreb_i1	subsystemA.coreb_i1.unique_core_rtl_tes		i9	i15				

```
// per ssh/icl pass/fail result for a pattern
foreach (var ssh in DeviceInfo.Pattern[ssnPatternSet.Value].IclInstances) {
    TheExec.Flow.TestLimit(ssh.FailedFlag, PinName: ssh.InstanceName,
ForceResults: tlLimitForceResults.Flow);
}
// per device core pass/fail result for binning
foreach (var core in DeviceInfo.Cores) {
    TheExec.Flow.TestLimit(core.TestResults(), PinName: core.Name,
ForceResults: tlLimitForceResults.Flow);
}
```

• Site Generic

Last but not least, the DeviceInfo object should do everything possible to comply to **Site Generic**. i.e. the pass/fail flag shall be `Site<bool>`, `.IsMasked` shall be `Site<bool>`... But for non-unique info like `InstanceName` and `PatternName`, they shall be `READONLY` strings or other generic types.

• Sample Code

```
// WORKING ZONE
```

Independent class per Flavor

As for now this will not be implemented since we belief using one class handling all three types will be more beneficial.

Debug Tools

Failing Core Identification

Currently there is no good debug tools support in IG-XL for SSN. Some features are planned in the future, but it has not decided yet, which and how they would look like. We will used CSRA to do some prototyping to bring in ideas we have in mind in order to do a proof of concept.

Core Status Display

The intention is to have a graphical way to show the status of each core and also allow to easily disable and enable cores.

Here are simple examples how we could show a high level summary for a single site. It is just to illustrate how the debug information could be shown, the naming of the differnt cell is not fix yet:

	Single Site						
	ICL Instance						
	Group A		Group B				
Pattern	A	B	C	D	E		
xxx	Pass	Fail	Pass	Fail	Pass		Disabled ICL Instance
yyy			Pass	Pass	Pass	Pass	Passing ICL Instance
zzz	Pass	Pass	Fail	Pass	Pass	Fail	Failing ICL Instance

as well as for multi site:

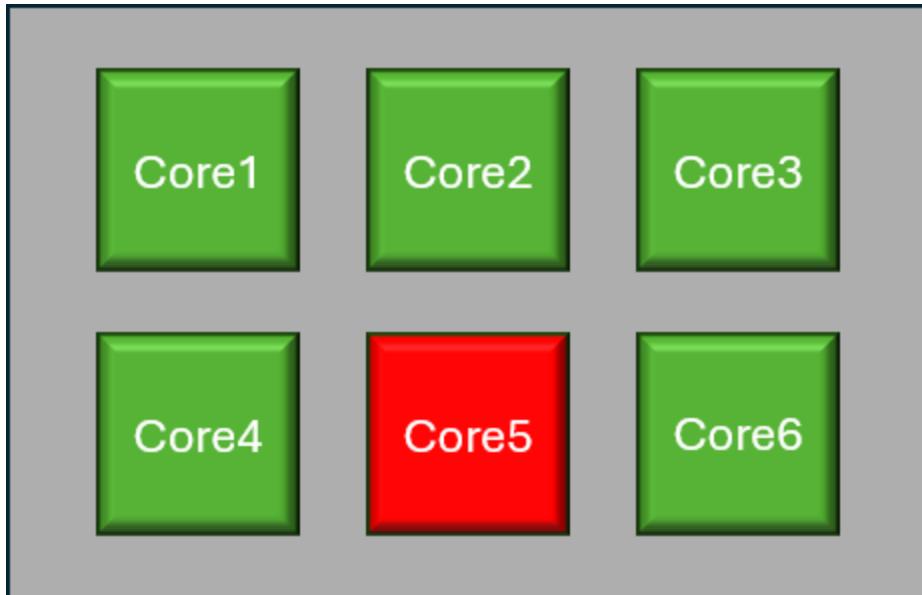
	Multi Site						
	ICL Instance						
	Group A		Group B				
Pattern	A	B	C	D	E		
xxx	Pass	50%	Pass	Fail	Pass		Disabled ICL Instance
yyy			Fail	Pass	Pass	50%	Site statistics
zzz	Pass	Pass	50%	Pass	Pass		

There some more ideas we have discussed, but we are currently not planning to implement them. We keep them here for documentation purposes. In addition more detailed information about failing cores could be provide in a result table:

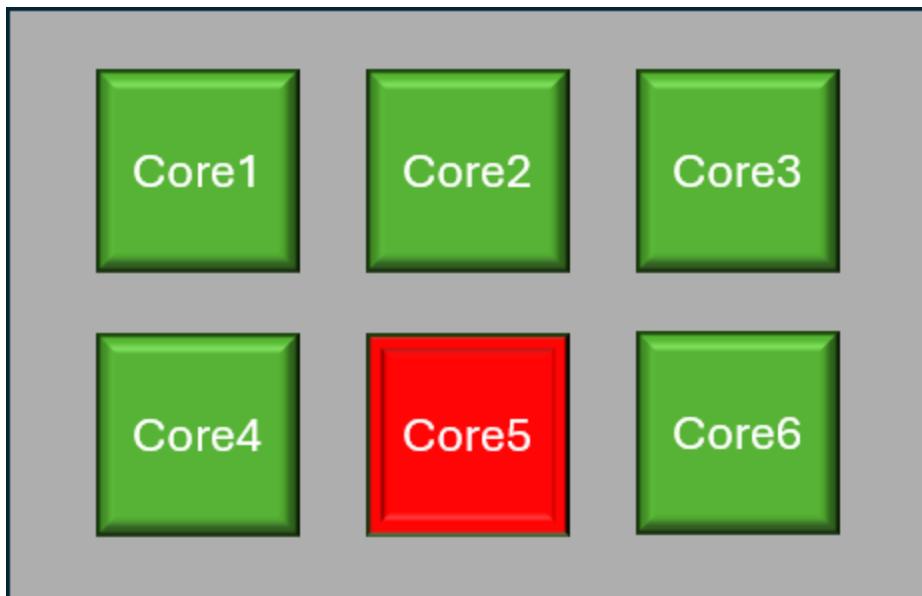
Failing Core Result Table					
Site	Pattern			CoreName	PinName
0	identical_cores_top_retargeting_stuck_serial_scan_ssn_patset_hi			subsystemB/coreb_i1	gpo_0
0	identical_cores_top_retargeting_stuck_serial_scan_ssn_patset_hi			subsystemB/coreb_i1	gpo_0
0	identical_cores_top_retargeting_stuck_serial_scan_ssn_patset_hi			subsystemB/coreb_i1	gpo_0
0	identical_cores_top_retargeting_stuck_serial_scan_ssn_patset_hi			subsystemB/coreb_i1	gpo_0
0	identical_cores_top_retargeting_stuck_serial_scan_ssn_patset_hi			subsystemB/coreb_i9	gpo_0
0	identical_cores_top_retargeting_stuck_serial_scan_ssn_patset_hi			subsystemB/coreb_i3	gpo_0

but this should be either replace by an graphical implementation or used to provide additional data.

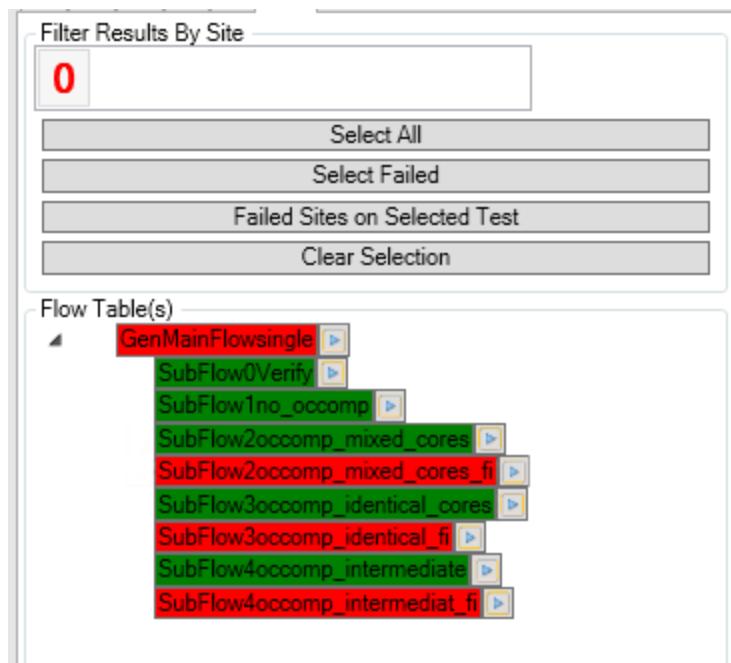
It would be also nice to show the core status, in this case, all cores are still enabled.



Show core status, similar to above, but now core 5 is disabled.



An alternative solution could be a debug display looking similar than the graphical results view in IG-XL.



Reference

- When dealing with ssn, Siemens created many new concepts along with a lot of new names/acronyms. To help user read this article easier, we also prepared a [Glossary for ssn](#).
- Another interesting difference(from traditional scan pattern) is how should a ssn scan pattern be executed and why. Here is another page that may help you get an answer: [ssn TestFlow](#)

Stepping

- XXX

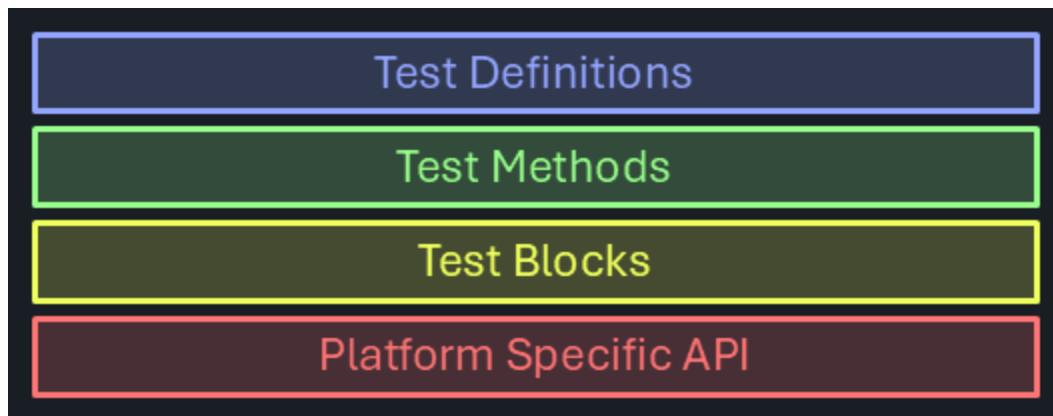
Test Abstraction

For very simple devices decades ago, implementing the test solution entirely with custom code programming hardware level directly was a reasonable approach. It provided good performance in a resource constrained environment at acceptable efforts.

That model doesn't scale - certainly not at the rate of how device complexity increases. Test solutions have become massive, and the effort to implement them is hardly manageable without a revised approach.

Modular concepts and code reuse can significantly reduce the overall effort, but requires the introduction of abstraction layers in a test solution stack-up.

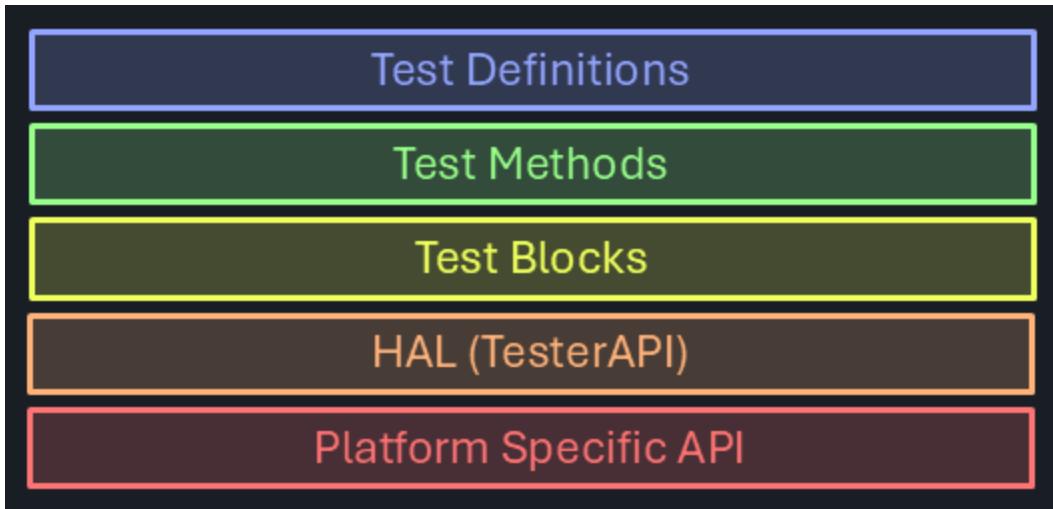
Test Solution Stack-Up



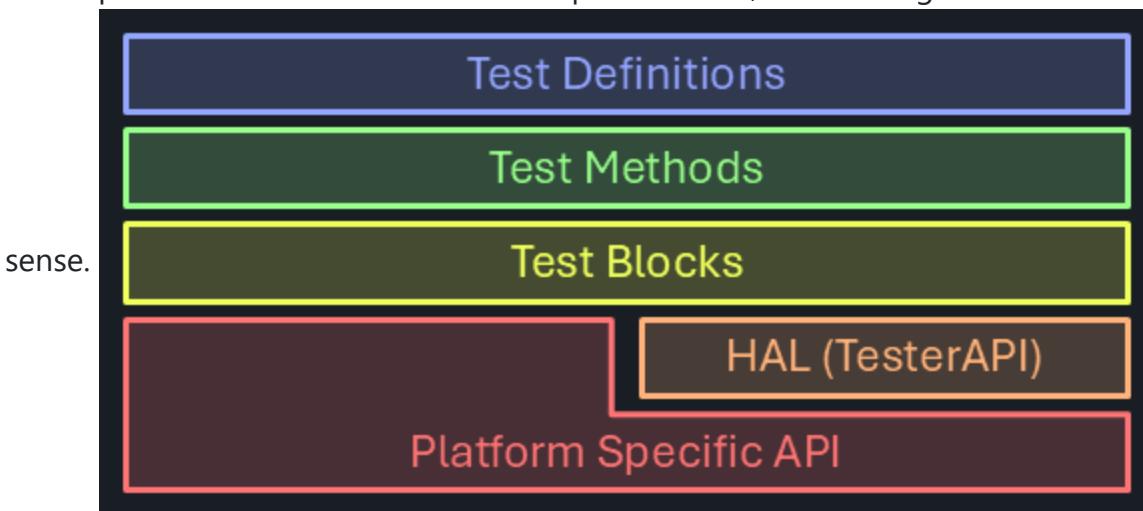
Platform Specific API

At the lowest level, the native programming interface of the tester hardware is located. It is typically very specific to a tester model, but allows accessing all features and unlocking inherent performance. It may not be easy to use or intuitive, and while test code directly using that may achieve great performance, but can be hard to create and maintain.

Specifically when [platform independence](#) is desired, inserting a hardware abstraction layer can be helpful. It would translate hardware specific calls into a generic interface that is available on multiple platforms.



More flexibility is possible when the hardware abstraction layer is not exclusively used - allowing access to unique architectural features for best performance, but utilizing the HAL in areas where that makes



Test Blocks ("*Tester Architecture Abstraction*")

The next level of abstraction is achieved by grouping hardware calls into higher level language constructs covering functional tester areas that are often used together. This can simplify common tasks on a specific tester, and also encapsulate a possibly beneficial execution order optimized for the intended use case.

Block abstraction allows for concise test code, with a focus on listing the logical steps required to perform a basic continuity test:

```
using static Demo.TestLib;

[TestMethod]
public void SimpleContinuity(string digPins, string powerPins, double forceCurrent) {

    // setup
    string allPins = Utils.MergePinLists(digPins, powerPins);
```

```

Connect(allPins);
Setup.ForceV(powerPins, 0 * V);
Setup.ForceI(digPins, forceCurrent, Measure.Voltage, 2 * V, forceCurrent);
Setup.Gate(allPins, true);

// measure
TheHdw.Wait(1 * ms);
PinSite<double> meas = Acquire.ReadMeter(digPins);

// reset
Setup.Gate(allPins, false);
DisConnect(allPins);

// datalog
Datalog.TestParametric(meas, "V");
}

```

NOTE

Note how the sequence of steps performed is highly visible with self-explaining syntax highlighting the intention ([Connect](#), [Setup](#), [Acquire](#), [Datalog](#), ...). The implementation is buried within these functions, which the user can see, step through or modify (copy & customize) if needed.

TIP

The PublicAPI call `TheHdw.Wait(1 * ms);` did not slip in by accident. If a call to wherever is suitable for the intended abstraction level, and it's purpose & functionality is apparent, there's nothing wrong with that.

Test Methods ("*Test Technique Abstraction*")

This level covers the implementations of test concepts, specific to certain device features or functional blocks. Test methods can be parameterized, but a tradeoff needs to be found between generic (many parameters, complicated, overhead) and specific (barely reusable, requires many of them, hard to maintain).

Test Methods should clearly reflect the high level information flow and functional logic of a test technique implementation. It's the level at which device and tester actions are engaged without going into the last technical detail. By covering the "how", the "what" becomes visible and comprehensible, for the code authors, the maintainers and the production engineers chasing yield problems.

To achieve that, the code needs to be brief, self explaining and provide a good overview of how a pass / fail decision is being made. A constant abstraction level is maintained, and jumps between high-level and detailed hardware calls is avoided.

Test Definitions ("*Test Implementation Abstraction*")

At the top level, test specifications defines what has to be tested, how that needs to be done, the consecutive order to execute and what to do with devices based on the results. If done formally, that information may be used by a program generator to automatically create, instruct or use the components underneath.

Ideally, test definitions don't contain any tester / platform dependence. They are generic and free from tester specific terminology.

Unit- & Integration Testing

In today's software development landscape, simply writing code isn't enough to meet customer expectations. Users rightfully demand that software is thoroughly verified before deployment. This mind-shift is driven by ever-increasing complexity and software savvy-ness of almost any product, and ATE systems specifically.

Without that, a reliable system is hard to achieve.

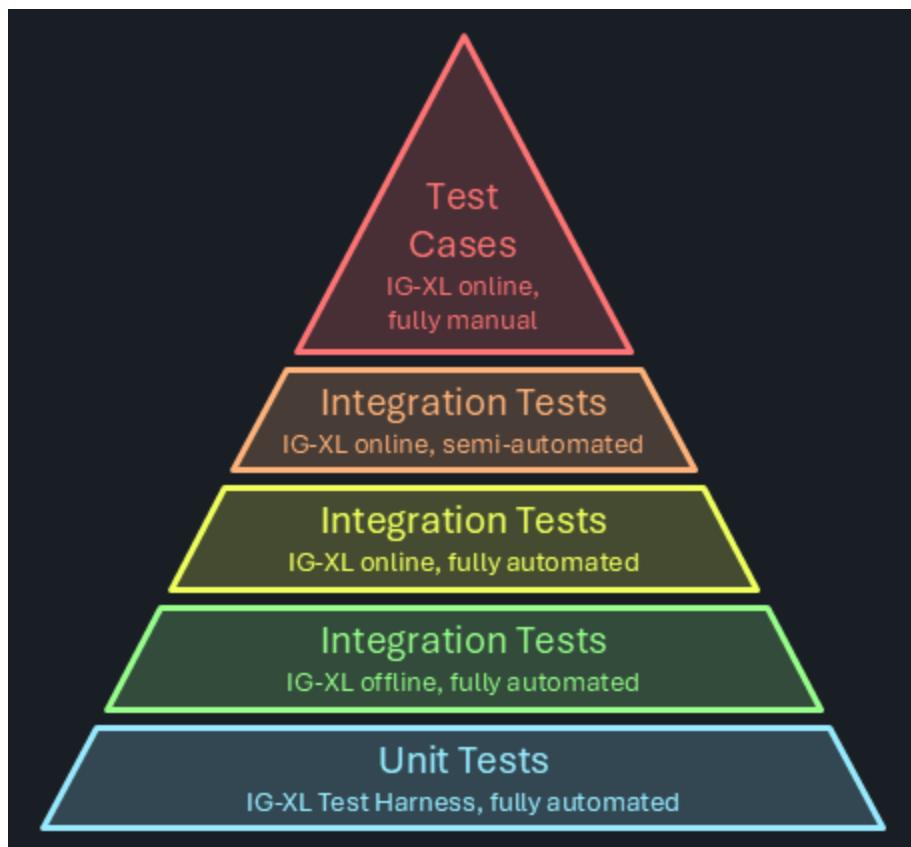
Objective

Within the C#RA project, the following goals need to be achieved:

- achieve great coverage (strive for 100%, make trade-offs only where not possible, not if "too much effort")
- provide earliest possible feedback to users
- provide an environment that makes writing and maintaining test efficient
- provide a test case framework that can easily be used to reproduce issues
- reproduce issues found in a unit or integration test before fixing it to confirm the fix is effective and recurrence is prevented

Test Strategies

Various options exist for software testing, and they come with pro's and con's.



Software testing doesn't come for free. As IG-XL prominently states "weeks of pure test runtime is performed for every build". That's achieved by massive parallelism of test hardware, so that the results are available in ~ 6 hours. Even that means that fails possibly requiring rework are often detected on the next working day, impacting schedule and adding friction when having to multiplex between tasks. The engineering effort to maintain all this is significant.

NOTE

Strictly typed languages (like C#) enable the compiler and the IDE to detect and flag code issues right at design time. That's the fastest and most efficient way to improve quality possible. Type mismatches, calls to non-existent functions, uninitialized objects, ambiguities and to some extent null reference errors are reliably prevented. The likelihood for code to run correctly at the first attempt is significantly improved.

There's always some cost, and proponents of loosely typed languages will cite a longer learning curve and higher efforts instead of "quickly get something done" as downsides. Especially for large projects with a focus on reliability these can be seen as investments that'll quickly reward.

Strictly typed languages help at the foundation of the pyramid by already removing the need for test coverage of many functional aspects of the software - it wouldn't be allowed to build if violated.

Unit Tests

In the pyramid above, the bottom is considered the preference. Unit testing means a focus on granular functional entities (=units) to be verified independently from a larger context or system where they would be embedded in.

Instead, the code is isolated and any dependencies are simulated. When using IG-XL Test Harness, test code **only thinks** it's talking to IG-XL's public API, but instead it's accessing a simplified copy (=mock) of that with little to no functionality. That makes the tests run faster on machines that don't even need IG-XL (and Excel and Oasis) installed.

By focusing on small & granular code entities (single methods, classes, types), it's easier to reach corner cases that'll utilize the different behaviors and allow comparing against expectations.

IMPORTANT

Try to achieve as much test coverage as possible with unit tests. It's the most efficient option.

IG-XL_TestHarness

[IG-XL Test-Harness](#) is a product, which mocks the IG-XL interfaces, so that test methods can be executed within MSTest, and no need to have IG-XL / Excel running or even installed. By simulating IG-XL's functionality, the Test-Harness enables the use of Test Explorer and makes 'dotnet test' functional, providing a seamless integration with the .NET testing tools.

Add details how to write unit tests here

Offline Integration Tests

Certain functionality requires IG-XL to be present, even an offline scenario can be sufficient. Examples are for instance where the mechanics of a test method can't be simulated without changing it.

Add details how to write offline tests here

Online Integration Tests

Some verification may require actual tester hardware or device responses. These tests have to be run asynchronously, because tester hardware isn't exclusively available.

The current plan is to have this executed daily on weekdays @ noon China time. The process is mostly automated, but requires a manual kick-off.

Add information about which can be automatic and which are manual, and how to write them

Online / Offline Test Cases

Some feature verification can't be executed via automation as it may be interruptive or require system or context changes that are either impossible or very difficult to do. These tests are done in test case workbooks, which users derive from the demo job based on ADU hardware.

Ideally, this category is empty.

Add details how to create test cases and if & where to maintain.

Implementation & Execution Matrix

Test type	Implemented in	Running on / at	Results will
Unit tests using IG-XL Test Harness	XXX project	GitHub cloud runner at every push & pull	block a merge on fails

Test type	Implemented in	Running on / at	Results will
		request	
Offline Integration Tests using IG-XL	YYY project	self hosted runner at every push & pull request	block a merge on fails
Online Integration Tests using IG-XL	ZZZ project	tester EV-??? / daily @ noon China time	be emailed? result in issues?
Online Integration Tests using IG-XL (semi-auto)	QQQ project	tester EV-??? / daily @ noon China time	be posted in DocFX pages? Manually processed?
Online Test cases	local copy of the demo workbook	authors discretion	be manually analyzed and captured as issues if needed

Alternatives Considered

AutoTest / AutoTestPro

[AutoTest](#) is an approach, developed as a proof-of-concept where IG-XL takes the lead in executing all the tests. In this setup, IG-XL is responsible for running the test code and logging the results to a log window. This approach does not integrate with MSTest or the Test Explorer, meaning that standard .NET testing tools, such as dotnet test, are not compatible with it. While AutoTest provides a way to execute tests directly within Excel, it lacks the integration with the broader .NET testing ecosystem.

[AutoTestPro](#) is a hybrid solution that combines the strengths of both the AutoTest and Test-Harness approaches. In this method, MSTest is used in conjunction with the Test Explorer to manage and display test results, while also starting and interacting with the Excel process to execute the test code.

This option was selected for the C#RA in its initial phase as it allows quick coverage ramp with foreseeable efforts, as that model had been used in other, large-scale projects before. After the IG-XL Test Harness product had further matured, the team switched over to that option. Migrating the existing tests was done in a consolidated, but not insignificant effort.

Transition phase

In the first phase of this project unit-tests were written with the **AutoTestPro** approach. Which was very successful to get the team going. Currently there are ~1000 unit-tests written for the **C#RA** project.

Meanwhile the **Test-Harness** product matured and the C#RA project will be one of the first Teradyne internal projects that will take advantage of it and be a tester of its capability and its use.

Because the previous approach needed a special infrastructure to work and the Test-Harness is much easier to use, a new structure for unit-tests needs to be implemented.

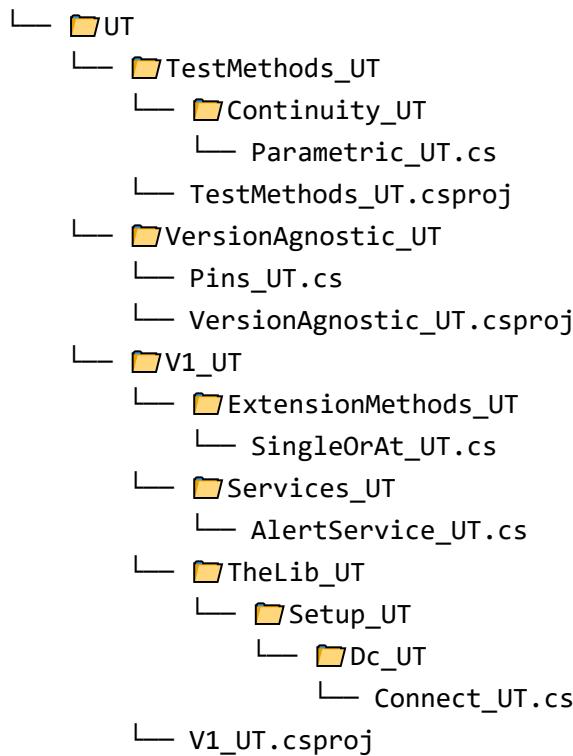
Architecture

The goal of unit-testing is that we aim 100 % line coverage for all future versions of C#RA. Therefore, we need a structure to easily maintain and extend it.

The **Test Explorer** of Visual Studio, is grouping unit-tests by **Project, Namespace, Class and Method**.

The [code-structure](#) shows that **Test Methods, Test Blocks, Services, Types, Extension Methods** are the things that need to be unit-tested. Test Methods and Types do not have a version, but Test Blocks, Services and Extension Methods are versioned.

To show the customer how to write unit-tests with Test-Harness and C#RA a unit-test project called **TestMethods_UT** will be introduced, that project only tackles unit-tests for **Test Methods**. A separate project is needed for **Types** and one project per version **Vx_UT**. All of those project will be placed in the **UT** folder inside of /src.



Historic Data

Problem Statement

The main issue is that the test code for IG-XL needs to be executed within an Excel process, but [MSTest](#), the testing framework, operates in a separate process. MSTest provides the test explorer to select single, or multiple tests to run, it also displays the results. Both processes are trying to control the execution, which creates a conflict.

Architecture

The architecture of this solution is designed to handle the complexities of integrating MSTest with Excel for unit testing. It involves multiple projects and shared components to ensure flexibility and efficiency in both single/multiple run and pipeline run workflows.

UnitTestSetup (shared)

has the Setup TestClass which does the communication between MSTest and IG-XL.

UnitTestBase (shared)

is the base class for every unit-test class that will be executed by IG-XL.

UnitTestPipeline (shared)

consists of one method, that will trigger IgxlTestHost to call every method via reflection that is a TestMethod.

UnitTest (shared)

is where every user will write the unit-tests.

MsTestHostPipeline (MSTest)

is a MSTest project that will call the single TestMethod from UnitTestPipeline. It imports the UnitTestPipeline, UnitTestBase and UnitTestSetup projects.

MsTestHost (MSTest)

is a MSTest project that gives you the ability to call each unit-test individually or in groups. It imports the UnitTest, UnitTestBase and UnitTestSetup projects.

IgxlTestHost (IG-XL)

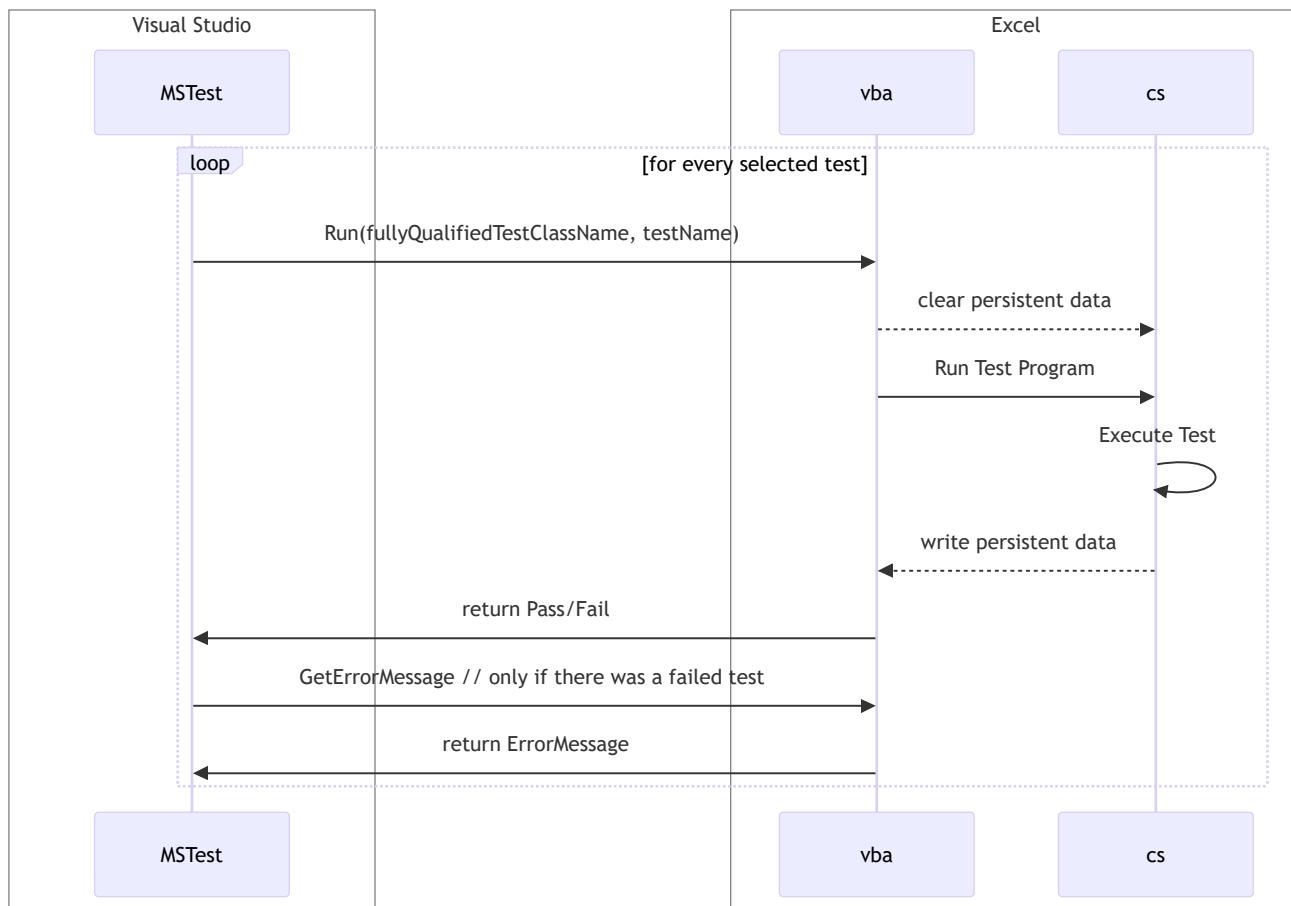
is an IG-XL project that has the infrastructure to call selected or all unit-test via reflection, depending on different aspects. It imports the UnitTest, UnitTestPipeline and UnitTestBase projects.

Workflow

In the context of integrating MSTest with Excel for unit testing, two distinct workflows have been developed to address different testing needs and optimize performance: the single/multiple run workflow and the pipeline run workflow. By having both workflows, you can choose the most appropriate method based on the specific requirements and context of your testing, ensuring both efficiency and reliability.

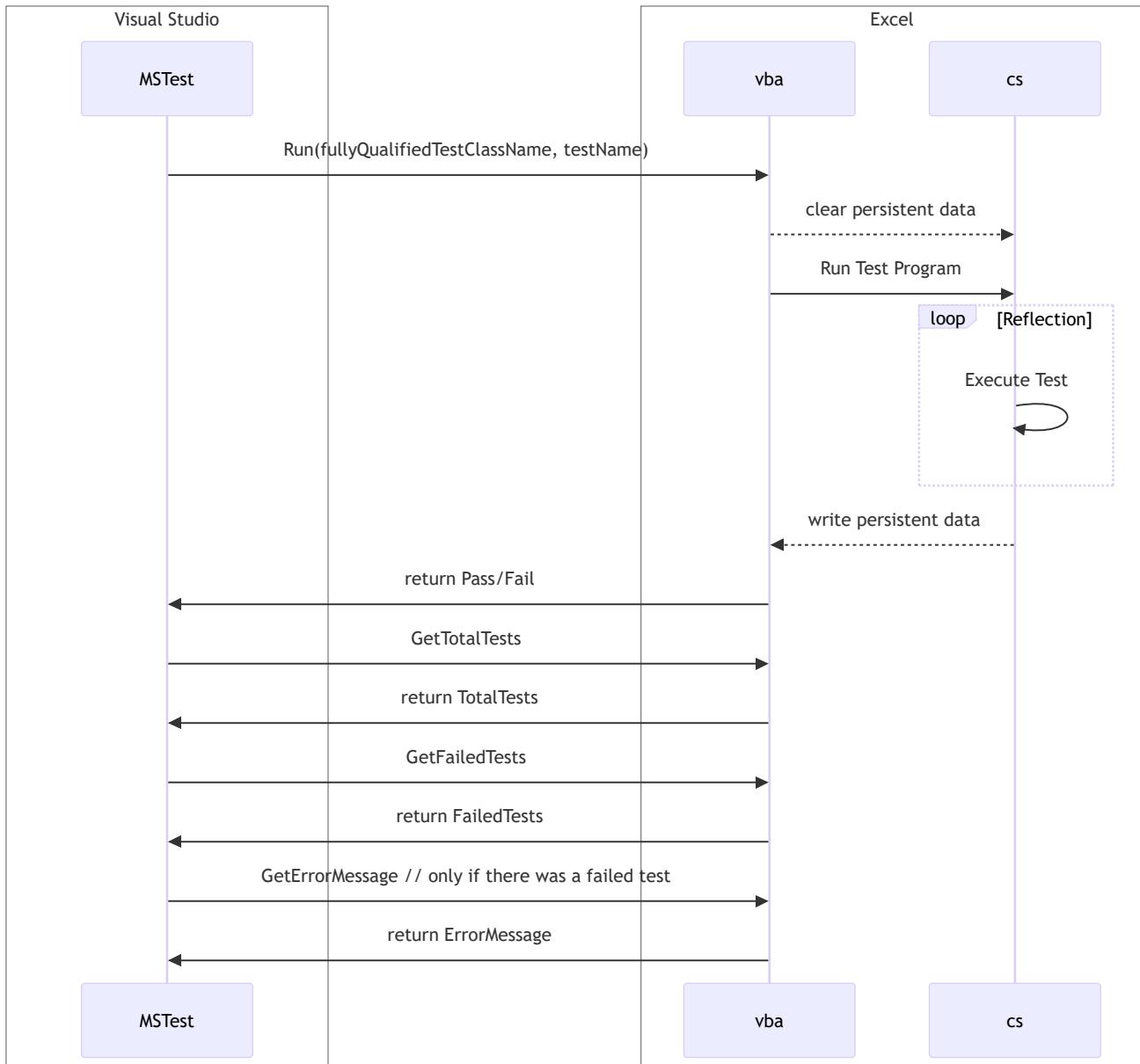
Single/Multiple Run

In this approach, the IG-XL flow executes a single test. This ensures that each test is executed in a fresh environment, which can be particularly useful for debugging or when running a small number of tests. However, this method introduces overhead, as the IG-XL flow needs to be initialized and terminated multiple times. While this overhead can be acceptable for smaller test sets, it may become inefficient for larger test suites and especially for pipeline runs.



Pipeline

In this approach, the IG-XL flow is started once at the beginning of the test run and remains active throughout the entire testing session. All tests are executed within this single instance of IG-XL, and the results are collected and returned to MSTest at the end. By reducing the overhead to a one-time cost, this workflow significantly improves efficiency, making it ideal for running a large number of tests or for automated testing environments.

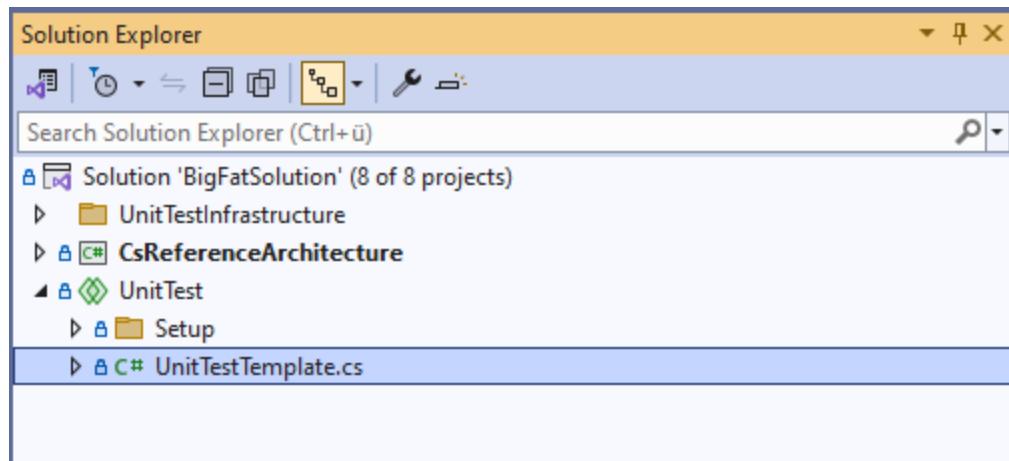


Use-Model

To utilize this architecture, users only need to focus on writing their unit tests within the UnitTest shared project. The process is straightforward and user-friendly:

Locate the Template Class

In the UnitTest shared project, there is a template class provided with simple test cases.



UnitTestTemplate.cs

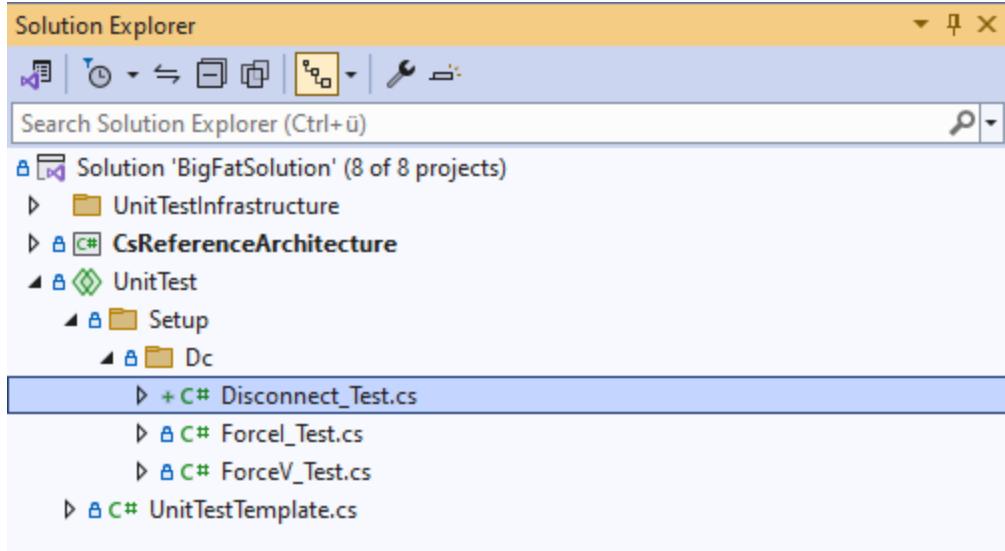
MsTestHost

```
1  using Teradyne.Igxl.Interfaces.Public;
2  using static UnitTest.Utilities;
3
4  #if IgxlTestHost
5  using IgxlTestHost_NET;
6  using MsTest = IgxlTestHost_NET.AutoTest.Attributes;
7  #else
8  using MsTest = Microsoft.VisualStudio.TestTools.UnitTesting;
9  using Microsoft.VisualStudio.TestTools.UnitTesting;
10 #endif
11
12 namespace UnitTest {
13     [MsTest.TestClass]
14     public class UnitTestTemplate : UnitTestBase {
15
16         [MsTest.TestMethod]
17         public void TestMethod1() {
18             if (IsNotExcel) return;
19             Site<int> a = new Site<int>(5);
20             Assert.AreEqual(5, a[0]);
21         }
22
23         [MsTest.DataTestMethod]
24         [MsTest.DataRow(5, 2, 7)]
25         [MsTest.DataRow(0, 5, 5)]
26         [MsTest.DataRow(0, 0, 0)]
27         [MsTest.DataRow(-10, 2, -8)]
28         public void DataTestMethod1(int valueA, int valueB, int expected) {
29             if (IsNotExcel) return;
30             Site<int> a = new Site<int>(valueA);
31             Site<int> b = new Site<int>(valueB);
32             Site<int> c = a + b;
33             Assert.AreEqual(expected, c[0]);
34         }
35     }
36 }
37 }
```

UnitTest.UnitTestTemplate

Copy and Paste

Users can copy and paste the entire template class into their own test file (e.g. 'Disconnect_Test.cs'). The file name should be the method name that you want to test with a specified postfix of '_Test'.



Change the namespace according to the hierarchy of the method you want to test. (e.g. 'TestLib.Setup.Dc.Disconnect()'). Also change the class name to match the file name (e.g. 'Disconnect_Test.cs').

```
1  using Teradyne.Igxl.Interfaces.Public;
2  using static UnitTest.Utilities;
3
4  #if IgxlTestHost
5  using IgxlTestHost_NET;
6  using MsTest = IgxlTestHost_NET.AutoTest.Attributes;
7  #else
8  using MsTest = Microsoft.VisualStudio.TestTools.UnitTesting;
9  using Microsoft.VisualStudio.TestTools.UnitTesting;
10 #endif
11
12 namespace UnitTest.Setup.Dc {
13     [MsTest.TestClass]
14     public class Disconnect_Test : UnitTestBase {
15
16         [MsTest.TestMethod]
17         public void TestMethod1() {
18             if (IsNotExcel) return;
19             Site<int> a = new Site<int>(5);
20             Assert.AreEqual(5, a[0]);
21         }
22
23         [MsTest.DataTestMethod]
24         [MsTest.DataRow(5, 2, 7)]
25         [MsTest.DataRow(0, 5, 5)]
26         [MsTest.DataRow(0, 0, 0)]
27         [MsTest.DataRow(-10, 2, -8)]
28         public void DataTestMethod1(int valueA, int valueB, int expected) {
29             if (IsNotExcel) return;
30             Site<int> a = new Site<int>(valueA);
31             Site<int> b = new Site<int>(valueB);
32             Site<int> c = a + b;
33             Assert.AreEqual(expected, c[0]);
34         }
35     }
36 }
37 }
```

Customize

Modify the copied template to suit your specific testing needs. Add or change test methods, inputs, and expected outcomes as required.

```
Disconnect_Test.cs  UnitTestTemplate.cs
MsTestHost
1  using Teradyne.Igxl.Interfaces.Public;
2  using static UnitTest.Utilities;
3  using Playground;
4  using static Teradyne.Igxl.Interfaces.Public.TestCodeBase;
5
6  #if IgxlTestHost
7  using IgxlTestHost_NET;
8  using MsTest = IgxlTestHost_NET.AutoTest.Attributes;
9  #else
10 using MsTest = Microsoft.VisualStudio.TestTools.UnitTesting;
11 using Microsoft.VisualStudio.TestTools.UnitTesting;
12 #endif
13
14 namespace UnitTest.Setup.Dc {
15     [MsTest.TestClass]
16     public class Disconnect_Test : UnitTestBase {
17
18         [MsTest.TestMethod]
19         public void HappyDay() {
20             if (IsNotExcel) return;
21             Pins pins = new Pins("dig,dsvi,dcvsl,dcvs2");
22             TestLib.Setup.Dc.Disconnect(pins, true);
23             Assert.AreEqual(tlOnOff.On, TheHdw.PPMU.Pins("dig").Gate);
24             Assert.AreEqual(tlDCVGate.GateOn, TheHdw.DCVI.Pins("dsvi").Gate);
25             Assert.AreEqual(true, TheHdw.DCVS.Pins("dcvsl").Gate);
26             Assert.AreEqual(true, TheHdw.DCVS.Pins("dcvs2").Gate);
27         }
28     }
29 }
30 }
```

This approach simplifies the process of writing unit tests, ensuring that users can quickly get started without needing to understand the underlying complexities of the architecture. By following these steps, users can efficiently create and manage their unit tests, leveraging the robust framework provided by the architecture.

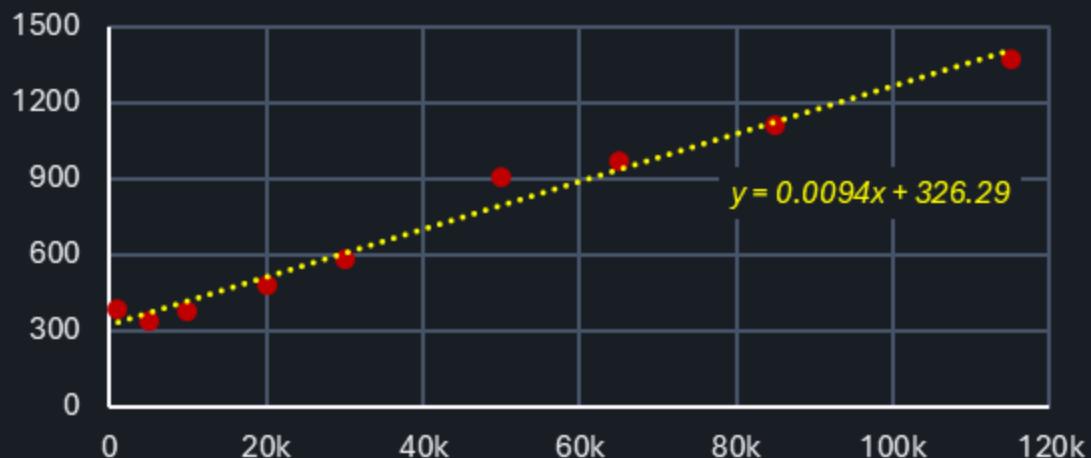
Performance

To ensure the robustness and efficiency of the architecture, extensive stress testing was conducted with varying sizes of unit tests. The tests included sets of 1k, 5k, 10k, 20k, 30k, 50k and 115k unit tests. The results demonstrated that the architecture could handle large volumes of tests efficiently.

There was a consistent overhead of approximately 300 seconds, regardless of the number of tests. This overhead is associated with the initialization and management of IG-XL.

The total execution time scales linearly with the test count. In this case, ~10ms were consumed per test, which can be considered to have realistic & representative complexity.

Test Execution Time [s] vs. Test Count



Backwards Compatibility

Compatibility is a sensitive topic for libraries that are widely used. The cost and effort to handle incompatible updates quickly scale with broad adoption, hence compatibility requirements need to be managed well.

Unfortunately, there is no perfect one-fits-all approach, and every solution comes with tradeoffs. Here's the model C#RA has selected, along with the logic chain why this is considered the best option.

Compatibility Philosophy

C#RA aims to minimize breaking changes while maintaining the flexibility to improve the library when necessary. The approach balances stability for existing users with the ability to deliver better solutions for new and evolving requirements.

Compatible Changes

The following types of changes are considered compatible and can be made in minor version updates:

- **Bug fixes** that correct unintended behavior without affecting legitimate use cases
- **Additional functionality** through new test blocks, test methods, or services
- **Additional overloads** for existing methods to support new use cases
- **Additional optional parameters** on existing methods with sensible defaults

These changes preserve existing code behavior and require no modifications to test programs using previous versions of the library.

Incompatible Changes

While the strong preference is to avoid them, incompatible changes may be necessary when:

- Original design choices prove inadequate for real-world requirements
- New requirements emerge that cannot be satisfied within the existing API structure
- Maintaining compatibility would compromise the quality or usability of the solution

When incompatible changes are required, they result in a new major version. The goal is to minimize such changes and introduce them only when the benefits clearly outweigh the migration effort for users.

Version Numbering

C#RA uses a **Major.Minor** version numbering scheme:

- **Minor version increments** (e.g., 1.0 → 1.1) indicate compatible additions: new features, bug fixes, and enhancements that do not break existing code.

- **Major version increments** (e.g., 1.5 → 2.0) indicate incompatible changes that may require code modifications in test programs.

Every release receives a version number. There are no unofficial versions, patches, or updates to existing releases. Each version is clearly identified through:

- Git tags in the repository
- Assembly attributes in the compiled code
- File names in release packages
- API methods that return the version information

Major version updates have no predefined schedule. They are introduced only when necessary, with the goal of keeping them rare.

Development Model

Development always targets the latest version of C#RA. There are no separate release branches, and features or fixes are not back-ported to previous releases.

This approach keeps the development effort focused and avoids the complexity of maintaining multiple active versions. Because C#RA is distributed as source code, users with specific requirements can apply changes to their local copy if needed.

Impact on Test Programs

The compatibility model is designed around how test programs actually use C#RA.

Version Independence

Test programs include a specific version of C#RA, typically the latest available when the project starts. Each test program operates independently with its own copy of the library.

There is no global installation of C#RA that affects multiple test programs. Different test programs can use different C#RA versions without conflict.

Update Strategy

Released test programs running in production do not require updates to C#RA as long as they function correctly. Updates can be considered when:

- The test program needs modification for other reasons
- New C#RA features would provide significant value
- Bug fixes in C#RA address issues affecting the test program

This means that incompatible C#RA updates have a limited scope. Only test programs that choose to upgrade are affected. Legacy programs continue to work with their embedded C#RA version.

Migration Effort

When a test program does upgrade to a major version with incompatible changes:

- The update is deliberate and controlled by the test program team
- Only the specific test program being updated is affected
- The scope of changes is typically smaller than in IG-XL platform updates
- Test program source code provides full visibility into required modifications

This makes the compatibility challenge more manageable compared to platform-level updates where all programs must adapt simultaneously.

Key Differences from the Previous Model

C#RA previously used a versioned interface approach where multiple API versions (V1, V2, etc.) existed side-by-side within the same release. This model has been replaced with the simpler approach described above.

What Remains the Same

- Version numbers (`Major.Minor`) continue to be used
- Major version increments still indicate incompatible changes
- Minor version increments still indicate compatible additions
- The commitment to minimize breaking changes continues

What Has Changed

The versioned interface indirection (V1, V2 namespaces) has been removed:

- **Previous approach:** Multiple API versions shipped together, accessed via `using Library_v1;` or `using Library_v2;`
- **Current approach:** Single API version per release, with incompatible changes delivered through new major versions

This change simplifies the library structure and reduces maintenance overhead without sacrificing compatibility for users. Test programs still work with their specific C#RA version, but the implementation is more straightforward.

Implications

- Users will encounter incompatible updates when upgrading to new major versions
- There will not be different release branches - development follows a single path forward

- The migration effort is manageable because test programs control when and if they upgrade
- The problem is less complex than IG-XL platform updates where all programs must adapt simultaneously

The new model trades the ability to run multiple API versions in a single program for simplicity in library structure and maintenance. Since test programs embed specific C#RA versions and rarely need to mix API generations, this tradeoff favors practical usability.

THE PREVIOUS MODEL

Client code that was written for a previous revision of the C#RA library will keep running, even if a newer version of the library is installed. This is guaranteed by design and confirmed through auto tests.

General Concept

- "language compatibility" - code that was written for a previous version keeps running
- "functional compatibility" - functionality is added, but not taken away, even in incompatible updates
- no "binary compatibility" exists - source code only
- use a `major.minor` version number scheme
- major version number is compatibility indicator
 - different major version => result of compatibility breaking need
 - same major version => backwards compatible
 - interface & functionally compatible
 - binary compatibility is not applicable - we don't ship assemblies
 - additional features may be added (optional args, overloads, new nodes, remove error messages)
- all previous versions are delivered in the product
 - test programs specify the version they use, so existing jobs keep running even with an updated, incompatible version
 - all versions are unit tested
- new feature developments typically go into latest version
 - can be down-ported if customers require and no compatibility impact
- bug fixes should be applied to all versions
- implementation code is shared as far as possible

Example Code

Consider the following library code:

```

namespace Library_v1 {

    public static class TheLib {

        public static void Feature1() => Library.Implementation.F1();
        public static void Feature2() => Library.Implementation.F2();
    }
}

```

It has two features implemented, exposed in the object tree structure under `TheLib`. Test code consuming those features would specify a using directive for `Library_v1` to directly access the entry point node:

```

using Library_v1; // place a using with the intended Library version for this
specific file

namespace TestProgram {

    [TestClass]
    public class TestClass {

        [TestMethod]
        public void TestMethod() {
            // normal access to the features in the version stated in using directive at
            the top of the file
            TheLib.Feature1();
        }
    }
}

```

Now, if an incompatible change becomes necessary, v2 of the library is created:

```

namespace Library_v2 {

    public static class TheLib {

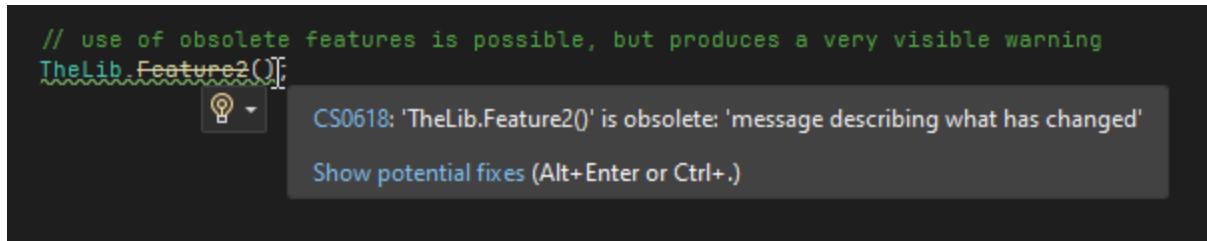
        public static void Feature1() => Library.Implementation.F1();
        public static void Feature2() => Library.Implementation.F2_new(); // this
replaces functionality that existed before
        public static void Feature3() => Library.Implementation.F3(); // this node is
added in V2
    }
}

```

This version provides a new & incompatible implementation of `Feature2` and adds an additional `Feature3`. The API for `Library_v2` is added in addition to the existing `Library_v1`. However, the now obsolete implementation of `Feature2` is marked with the `[Obsolete]` attribute:

```
namespace Library_v1 {  
  
    public static class TheLib {  
  
        public static void Feature1() => Library.Implementation.F1();  
        [Obsolete("message describing what has changed")] // nodes which receive  
        incompatible updates in newer releases are marked [Obsolete] in previous ones  
        public static void Feature2() => Library.Implementation.F2();  
    }  
}
```

It's a gentle reminder to avoid this functionality, because it has been replaced with a better implementation for a reason. `Feature2` can still be used, but the Visual Studio IDE provides clear feedback with a strike-through marking and a compiler warning:



At this point, it's a decision for the consumer of the library to determine the risk for new or existing code. The supplier of the library should clearly indicate > support policies for obsolete interfaces, giving the consumers enough time to react.

Feature implementations from other major versions available in the package, but not referenced in the file header `using` statements can easily be accessed through > their fully qualified name:

```
using Library_v1; // place a using with the intended Library version for this  
specific file  
  
namespace TestProgram {  
  
    [TestClass]  
    public class TestClass {  
  
        [TestMethod]  
        public void TestMethod() {  
            // normal access to the features in the version stated in using directive at
```

```

the top of the file
    TheLib.Feature1();

    // other versions can be accessed by using fully qualified name
    Library_v2.TheLib.Feature3();
}
}
}

```

Despite the API versioning requiring code duplication, implementation can still share the same code. In fact, by separating both, it's easy to support reuse and modular use of common functionality:

```

public static class Implementation {
    public static void F1() { }
    public static void F2() { }
    public static void F2_new() { }
    public static void F3() { }
}

```

Specific Implementation

Applying this versioning concept to the C# Reference Architecture requires careful design for intuitive use and bug-free implementation. Any levels of complexity and indirection added must be vetted for the costs and benefits actually added.

The following strategies are followed for the C#RA components:

C#RA Entity	Compatibility Strategy
Test Blocks	✓ Use versioned interfaces and are exclusively accessed via these (including internal access). Previous versions remain accessible.
Test Methods	✗ Don't offer versioning, releases contain the latest and greatest. Users can copy previous versions into custom user code.
Services	✓ Use versioned interfaces and are exclusively accessed via these (including internal access). Previous versions remain accessible.
Public Types	✗ Don't offer versioning, careful initial design and extension only updates maintain backwards compatibility.

C#RA Entity	Compatibility Strategy
Public Enums	✓ Use versioned versions mapping to internal, implementation supersets. Previous versions remain accessible.

Alternatives considered

Explicit `Vlatest`

It was considered to offer a dedicated latest version in addition to explicit `V1`, `V2` ... so that test blocks and test methods can opt to follow the latest and greatest per version. However:

- C# does not allow a way to serve interfaces in two namespaces, so this would result in yet another copy of all interfaces to be maintained, unit tested, and carried along.
- The C#RA provided test methods and test blocks showcase recommended practice as role model for users' custom implementations. Because of the strong motivation to keep our code up-to-date and use the latest features, the use of `Vlatest` would make sense - any functional issues resulting from incompatible changes could and would have to be addressed before release.

Broad use of `Vlatest` however would suggest this use model also for customers leaning their designs on the reference code. Then, their designs would also be susceptible to breaking for major version upgrading, so they are forced to edit code or resign from updating. Which was exactly what the introduction of versioned interfaces was trying to avoid.

The concept of offering an explicit `Vlatest` is not pursued.

Training Page

Welcome to the training page for C#RA!

 **NOTE**

 **This page is a placeholder while we work on the training materials.**

v0.13 - 2025-11-14

Breaking Changes

Removes Version-Specific Namespace "V1"

Simplify the API structure and debuggability by removing versioning.

Impact: User need to change their usings from `using Csra.V1;` and `using static Csra.V1.Api;` to `using Csra;` and `using static Csra.Api;`

Introduce a New Node Services

Instead of having all Services directly accessible from `Csra.Api`, they will be bundled together to `Csra.Api.Services`.

Impact: User need to change their usage of any Services to go through the new language.

```
// previously
SetupService.Apply("abc");
// new
Services.Setup.Apply("abc");
```

Migrate `AlertService.ValidationFail()` into `AlertService.Error()`

In cases where it's not clear whether code is being executed in a validation or a program run context, selecting the correct way to handle errors has been challenging because `AlertService` offered two distinct methods. To simplify this, and to prevent accidental use in the wrong place, the functionality has been merged in a common method that can be used anywhere and will internally decide on the correct handling.

```
internal static bool IsTrue(bool condition, string problemReasonResolutionMessage, string argumentName) {
    GetArgumentContext(argumentName, out int argumentIndex, out string messagePrefix);
    if (condition == false) {
        if (string.IsNullOrWhiteSpace(problemReasonResolutionMessage)) {
            AlertService.Error($"{messagePrefix}Provided 'ProblemReasonResolutionMessage' is null.", argumentIndex);
            return false;
        }
        AlertService.Error($"{messagePrefix}{problemReasonResolutionMessage}", argumentIndex);
        return false;
    }
}
```

```
    }  
    return true;  
}
```

Impact: Improved use model - flagging erroneous conditions to IG-XL is now uniform for runtime and validation.

Bug Fixes

Fix run-time-error in demo projects

Due to UltraFLEX only call in OnProgramLoaded, DemoCSRA and DemoCS caused a run-time-error if only UltraFLEXplus installed. By removing the unnecessary call, the error no longer comes up.

```
// Showcase how to use it in code, vs previous if possible.
```

No more errors when starting the demo programs.

Improvements

Clean up Demo Program Flows

- add test group separators
- align test group sequence
- update test numbers
- remove binning - not used in automation

Impact: Clear structure and comparability between C#RA, C# and VBT code.

Fix voltage range handling in `SetForceAndMeter`

Since ApplyLevelsTiming does not configure the voltage range of the power supplies, the voltage range should only be modified when necessary, using the `Modify` test block.

Impact: The voltage range will no longer be modified by the user when using method `SetForceAndMeter` in `forceVoltage` mode for DCVI and DCSV instruments. Adjustment of this parameter will be performed exclusively through dedicated configuration blocks, such as `Modify`, or by using the advanced `forceV` method.

Show list of valid options in case of failing enum argument validation

Enum values have to be provided as string into test methods. During validation, these strings are parsed into enum values. In case of a fail, now the error message shows all valid options.

```
TheLib.Validate.Enum(inputValue, nameof(inputValue), out TLibOutputMode enumValue);  
  
// Argument 'inputValue': Provided value '{inputValue}' could not be found within provided  
Enum 'TLibOutputMode' - the valid options are 'ForceVoltage, ForceCurrent, HighImpedance'.
```

Impact: Improved usability for enum type test method arguments.

v0.12 - 2025-10-17

Improvements

Restructuring Documentation

Removing and rearranging information to improve readability and orientation.

Offline Documentation Appearance Changed

Instead of shipping html files, a pdf file is being shipped as offline documentation.

v0.11 - 2025-10-10

Breaking Changes

Move TestMethods from Csra to Demo_CSRA

Demonstrate customer code utilizing Csra TestBlocks for reference and implementation guidance.

Impact: Needs to reference the [Demo_CSRA](#) instead of [Csra](#) on the reference sheet.

Bug Fixes

Fix Debug Experience

v0.10 had introduced a bug that users cannot debug C#RA code anymore. This is now fixed.

v0.10 - 2025-10-02

Breaking Changes

Dropping 10.60.xx Support

The C#RA isn't supporting any 10.60.xx version of IG-XL.

Impact: Customer needs to update to at least 11.00.00.

AlertService.ValidationFail Use-Model Change

Instead of inserting the `argumentName` as string, the `argumentIndex` is now needed to mark in IG-XL the offending column.

```
// previously
public void ValidationFail(string failMessage, string argumentName = "");
// new
public void ValidationFail(string failMessage, int argumentIndex);
```

Impact: Refactor use of AlertService.ValidationFail.

Bug Fixes

Missing Nuget Package

The `v0.9` of C#RA package had an issue with a missing nuget package. This has been fixed with conditional loading of that particular nuget package.

Improvements

New SDK-project format

The C#RA project is now using the new SDK format.

Impact: Users who use IG-Link to generate C#RA projects must ensure they are running Oasis version 4.5.10501 or higher.

Supporting IG-XL 11.00.01 Patch

The C#RA-Team is excited to announce that `Csra` now officially supports the IG-XL 11.00.01 Patch!

v0.9 - 2025-09-26

Breaking Changes

Signature of TestMethod Functional StaticPattern Changed

A boolean argument has been added to parametrize the datalog capability.

Impact: Customer may need to check their InstanceSheet to add a boolean to the arguments list.

Remove Static TryParse Method from Pins Object

The feature `Pins.TryParse` has been removed because the use-case has been fixed in another way.

Impact: Customer may need to remove the method from their TestProgram.

Move Programmatic Version Readout to Proper Location and Add XML Comments

The feature is now accessible via `Csra.V1.Api.Info.Version` following the scheme of the other C#RA features

Remove `SplitMultiCondition()` from `TheLib.Validate`

`SplitMultiCondition()` from Validation TestBlock has been replaced by Validation method '`MultiCondition()`'

```
// previously  
TheLib.Validate.SplitMultiCondition();  
// now  
TheLib.Validate.MultiCondition();
```

Impact: Refactoring needed, replace old with new method.

Bug Fixes

Bug Fix in Search Parametric

In the `Search Parametric` TestMethods, the voltage range for the forcing pin was not defined. The current update resolves this issue by using the `ForceV` method, which allows setting the voltage range.

Fix Bug Related to Measurement Pin Configuration

The bug related to measurement pin configuration has been resolved. The `SetForceAndMeter` method was replaced with `ForceHiZ` and `SetMeter`, eliminating the possibility of forcing a value on the pin and

ensuring correct measurement setup.

The test description has been adjusted to provide clarity regarding its use for voltage forcing and measurement.

Fix Bug Related to Gate Switching in Voltage Mode for the PPMU

The issue where the gate was only switching in current mode has been fixed, voltage mode is now correctly handled.

Fix Bug Related to LinearStopFromToCount and LinearFull Process

Comparing both algorithms with the same data previously resulted in different return values. This has been fixed, and the results are now identical for the same data.

Improvements

Add Parameter Validation for TestMethods

Validation Blocks have been added to the TestMethods, to verify the input arguments from IG-XL.

Impact: Expands argument validation coverage within C#RA.

LICENSE.txt has been Added to the Release Package

A license file is now part of the release package.

Add GreaterThan() & LessThan() Validation TestBlock

TestBlocks has been added to the Validation TestBlock family.

```
TheLib.Validate.GreaterThan(forceValue, 0, nameof(forceValue));  
TheLib.Validate.LessThan(forceValue, 6.5, nameof(forceValue));
```

Impact: Provides option to test if a value is greater/less than boundary.

Clean up Code Structures Corresponding to VBA's With Statements

Per recommended practice, temporary variables with a brief name should be used

```
{  
    var dcvs = TheHdw.DCVS.Pins(powerPin);
```

```
dcvs.Gate = false;  
dcvs.Disconnect(tlDCVSConnectWhat.Default);  
}
```

Impact: Functionally identical, readability improved.

Supporting IG-XL 11.00 Release

The C#RA-Team is excited to announce that [Csra](#) now officially supports the IG-XL 11.00 Release!

v0.8 - 2025-09-01

Breaking Changes

Renamed `TheLib.Validation` to `TheLib.Validate`.

```
// previously
TheLib.Validation.Pins(readPins, out _pins, nameof(readPins));
TheLib.Validation.Pattern(pattern, out _patternInfo, nameof(pattern));
TheLib.Validation.GreaterOrEqual(startIndex, 1, nameof(startIndex));

// now
TheLib.Validate.Pins(readPins, out _pins, nameof(readPins));
TheLib.Validate.Pattern(pattern, out _patternInfo, nameof(pattern));
TheLib.Validate.GreaterOrEqual(startIndex, 1, nameof(startIndex));
```

Impact: Potential renaming necessary.

Bug

Fixing the bug in Acquire.Measure

Fix the failure of the `.AddRange` method (for `PinSite<Samples<double>>`) by sequentially iterating through each measurement and reordering them based on the pin order defined in `Pins`  a temporary solution until `.AddRange` can be used.

Improvements

Add `ExtractRange()` feature to `Pins` class

Retrieve a subset of (consecutive) pins from the collection.

```
Pins _pins = new("pinA, pinB, pinC");
Pins theFirst= Pins(pins).ExtractRange(0, 1);
```

Users can now use index based subsets or implement serial operations, iterating through all available pins.

Added Argument Validation Test blocks

Created argument validation test blocks in line with definition(s) within the C#RA documentation.

Available argument validation methods are:

InRange - Checks if a numeric value is between two bounds.

GreaterOrEqual - Checks if a numeric value is greater or equal to a bound.

LessOrEqual - Checks if a numeric value is less or equal to bound.

Pattern - Checks for valid pattern spec and creates PatternInfo() object.

Pins - Checks for valid pin spec and creates Pins() object.

MethodDelegate - Checks for valid method spec and creates delegate object.

MultiCondition - checks for multicondition validity and creates the data array.

Enum - Checks string against provided Enum and returns enum value if applicable.

```
TheLib.Validate.InRange(clampVoltage, -2.0, 6.0, nameof(clampVoltage));
TheLib.Validate.GreaterOrEqual(clampVoltage, threshold, nameof(clampVoltage));
TheLib.Validate.LessOrEqual(clampVoltage, threshold, nameof(clampVoltage));

TheLib.Validate.Pattern(Pattern, nameof(Pattern), out PatternInfo patternInfo);
TheLib.Validate.Enum(stringEnum, nameof(stringEnum), out enum enumValue);
TheLib.Validate.Pins(pinList, nameof(pinList), out Pins pins);

TheLib.Validate.MultiCondition(string values, s => type.Parse(s), nameof(values), out type[] conditions, int? numConditions);
TheLib.Validate.MethodHandle(fullyQualifiedMethodName, nameof(fullyQualifiedMethodName), out MethodHandle<T> method);
```

Impact: This will allow for the user to deploy argument validation on Common test methods, resulting in a much quicker way of identifying invalid arguments.

Added Fail() Validation block

Fail() validation block is intended to raise an unconditional validation error when called.

```
TheLib.Validate.Fail(problemReasonResolutionMethod, nameof(Argument));
```

Impact: Provides a method that users can utilize when working with arguments that do not have existing validation methods.

Added IsTrue() to available Argument Validation Methods

IsTrue() is a generic validation check that can't be connected to a single parameter only (like illegal combination of parameters).

```
TheLib.Validate.IsTrue(condition, problemReasonResolutionMessage, nameof(condition));
```

Impact: This is meant to be a fallback for ANY checks. Allows for argument validation where the check cannot be connected to a single parameter.

Generate Visual Studio Solution file for the Demo project

Added a new command script [_LoadDemoProgram.cmd](#) that generates a Visual Studio solution file [Demo_Generated.sln](#) in the Demo directory.

[_LoadDemoProgram.cmd](#)

Implement layer for multiple transaction handlers

TransactionService now allows to add multiple transaction handlers and to switch usage between them.

Currently no impact for the user, as this is just the layer without functionality.

Implemented empty TransactionService methods

Implementation of all methods for future use in TransactionService. Currently only the methods without functionality exist, this will be implemented later.

N/A - examples will be available in [future](#) merges when functionality is implemented.

No impact for the user in this early implementation phase.

Remove workaround for FunctionalTestLimit bug

FunctionalTestLimit threw an exception if multiple sites and usage of site generics. As workaround conversion to SiteBoolean was implemented. This was fixed in IG-XL V11 Alpha Kit, thus workaround was removed.

N/A

There is no impact for the user, no changes in the API.

Implementation of Continuity.Parametric.Serial

A testing method is added that performs continuity testing on pins sequentially. The branch includes the test method, unit tests, and documentation.

v0.7 - 2025-08-01

Breaking Changes

BinarySearch test blocks changed, now dedicated implementations for `double` and `int`.

Different strategies mandated going away from generic implementations and making them type specific again. Now, there's an option for floating point and fixed point each. The floating point is implemented for `double`, correctly handles corner cases and is optimized for performance and memory footprint. The fixed point option uses `int` and correctly handles scaling the search intervals towards the (fixed point) resolution.

Impact: Users may review and update their implementations.

Remove RespectSettlingTime Argument

RespectSettlingTime parameter of all features was removed, as instruments already do a settlewait after applying settings. Users may need to adjust their code to get it working again.

```
// Old
dcvisetup.Add(new Setting.TheHdw.Dcvci.Pins.ComplianceRange_Positive(0.01, "dcvi1,
dcvi2", true));
// New
dcvisetup.Add(new Setting.TheHdw.Dcvci.Pins.ComplianceRange_Positive(0.01, "dcvi1, dcvii2"));
```

This change may improve testtime.

Support Update: IG-XL Version

Support for **IG-XL CI3 Kits (11.00)** has been discontinued. **IG-XL Kit 11.00 Alpha 4** is now the officially supported release. This version introduces a more stable foundation for ongoing development and testing.

Refactor PatternInfo-object

`Name` property is now readonly. Change setter of `ThreadingEnabled` from private to public.

Rename Instrument Type

Renamed UP5000 -> UPHP.

Impact: Potential renaming necessary.

Substitute `GetDelegate` `TestBlock` with `MethodHandle` Type

```
// previously  
TheLib.Validation.GetDelegate<T>("methodName");  
  
// now  
new MethodHandle<T>("FullyQualifiedName");
```

Impact: Substitute every use of `GetDelegate` with `MethodHandle`.

Bug Fixes

Fixing the bug in `SetForceAndMeter`

When calling `SetForceAndMeter` with the `Measure.Voltage` measurement configuration, the user will enter a voltage value for the `measureRange` variable. In the `ppmu.ForceVMeasureV()` configuration for PPMU, `measureRange` was previously used as the input, however, the method requires `MeasureCurrentRange`. Therefore, in this case, the setting was updated to use `ppmu.MeasureCurrentRange.Max`

The configuration for `MeasureCurrentRange` is still available through method `Modify()`.

Implementation changes for PPMU in `Modify`

The current changes allow parameters such as `voltage` or `current` to be set for PPMU without requiring the value for `mode` or `meterMode`.

The user is not affected by these changes, the function call remains the same.

Remove workaround for IG-XL bug

SetupService setups can now be created/added in `OnProgramValidated` if IG-XL version is 11.

Usage with IG-XL 10.60.01:

```
[TestMethod]  
public void SetupLoad() {  
  
    SetupService.Reset();  
  
    Setup initLeakage = new Setup("InitLeakageTest");  
    initLeakage.Add(new Setting.TheHdw.Digital.Pins.InitState(ChInitState.Hi,  
"nLEAB, nOEAB"));  
    initLeakage.Add(new Setting.TheHdw.Digital.Pins.InitState(ChInitState.Lo,  
"nLEBA, nOEBA"));  
    initLeakage.Add(new Setting.TheHdw.Digital.Pins.InitState(ChInitState.Off,
```

```

    "porta"));
        SetupService.Add(initLeakage);
    }
}

```

Usage with IG-XL 11.00.00:

```

[ExecInterpose_OnProgramValidated(255)]
public static void OnProgramValidated() {
    Setup initLeakage = new Setup("InitLeakageTest");
    initLeakage.Add(new Setting.TheHdw.Digital.Pins.InitState(ChInitState.Hi, new
List<string> { "nLEAB", "nOEAB" }));
    initLeakage.Add(new Setting.TheHdw.Digital.Pins.InitState(ChInitState.Lo, new
List<string> { "nLEBA", "nOEBA" }));
    initLeakage.Add(new Setting.TheHdw.Digital.Pins.InitState(ChInitState.Off, new
List<string> { "porta" }));
    SetupService.Add(initLeakage);
}

```

Impact: Move SetupService setup from flow to validation.

Improvements

Add offline passing mode to `TestParametric` blocks

Parametric tests can now be forced to pass offline, by datalogging values mid-way between the specified test limits. Any measurements or calculation results fed in are (offline only!) ignored and substituted with values that will never fail. A `BehaviorService` feature has been added to allowing opt-in to this new mode.

```

BehaviorService.SetFeature("Datalog.Parametric.OfflinePassResults", true);
TheLib.Datalog.TestParametric(result, 3 * V, "V");

```

Impact: Users can opt-in to this new use model, allowing flow path mechanics that are closer to reality.

1. Add 6 digital trim tests:

- `Trim.Digital.LinearStopTrip`: Use linear stop search with trip criteria to find where the test pattern result change from fail to pass.
- `Trim.Digital.LinearStopTarget`: Use linear stop search with target value to find which test step's output is closest to the target by capturing HRAM data.
- `Trim.Digital.LinearFullTrip`: Use linear full search with trip criteria to find where the test pattern result change from fail to pass.

- `Trim.Digital.LinearFullTarget`: Use linear full search with target value to find which test step's output is closest to the target by capturing HRAM data.
- `Trim.Digital.BinaryTrip`: Use binary search with trip criteria to find where the test pattern result change from fail to pass.
- `Trim.Digital.BinaryTarget`: Use binary search with target value to find which test step's output is closest to the target by capturing HRAM data.

2. Modify `LinearStopFromToCount`:

Change `LinearStopFromToCount` to compatible with `Site<int>` when using the `target` parameter.

Impact: No impact.

Add `BehaviorService`

`BehaviorService` manages configurable, program-wide behavior settings - allowing test programs to adapt to different user preferences or operational modes without modifying source code.

```
BehaviorService.SetFeature("MaxRetries", 5);
BehaviorService.SetFeature("Timeout", 30.0);
BehaviorService.SetFeature("EnableLogging", true);
BehaviorService.SetFeature("ApiEndpoint", "https://api.example.com");
```

Impact: Users benefit from a single, standardized way to address a common need, that is typically solved by a variety of incompatible strategies scattered throughout the test program.

Add `Pins.ArrangePinSite<T>()`

Combines multiple `PinSite<T>` instances into a single `PinSite<T>` by following this `Pins` object's sequence. Excessive objects are quietly ignored, missing ones result in a runtime exception.

```
PinSite<int> one = new PinSite<int>(["dig, dcvi"]);
PinSite<int> two = new PinSite<int>(["dcvs1, dcvs2"]);
Pins pins = new("dig, dcvs1, dcvi, dcvs2");

PinSite<int> arranged = pins.ArrangePinSite([one, two]);
```

Impact: Removes the limitation where measurement & readback test blocks were limited to a single instrument type only.

Add BinarySearch test blocks for integer DUT inputs

Binary searches for floating point and integer inputs require slightly different treatment to avoid rounding issues and corner cases. This task adds `int` type searches to the existing ones for `double`,

following the exact same use model and method signature.

```
Site<int> result = TheLib.Acquire.Search.BinarySearch(0, 100, inMinDelta, false,  
NonInverting, v => v >= outTarget, -9999, out var outResult);
```

Impact: Users now have tailored and optimized binary search algorithms for `int` and `double` inputs.

Add C#RA version readout API

Users can now read the C#RA version via language in code.

```
Console.WriteLine(Csra.Info.Version);
```

Impact: Improved maintainability and version tracking capability.

Add conditional PortBridge initialization to Demo program

Impact: Users can optionally run the Demo program in a scenario with or without PortBridge.

Add PortBridge library integration

PortBridge comes as a separate product that can optionally be used in IG-XL based test programs. C#RA supports scenarios with or without this library. In this first step the mechanics of a conditional reference is added, along with a compiler directive to mask calls so that they don't prevent builds when the library isn't available.

```
#if PORTBRIDGE_ENABLED  
    /// do fancy PortBridge stuff  
#endif
```

Impact: C#RA authors and users get conditional access to PortBridge language features.

Change `steps` notation to `count` in LinearSearch test blocks

The term `steps` is indicative for intervals, whereas the way it's used in the linear search test blocks it's being used for the number of overall measurements taken (`count`). The name change intends to remove ambiguity users have reported - note that no functional change applies. Also see

https://en.wikipedia.org/wiki/Off-by-one_error

```
LinearFullFromToCount<Tin>(Tin inFrom, Tin inTo, int inCount, Action<Tin> oneMeasurement)
```

Impact: Improved clarity for users when requesting step counts in linear search test blocks.

Change the input parameter of the Frequency.Timing() test method

Adjust the input parameters of the `Frequency.Timing()` test method to align it with other test methods, and allow IG-XL to verify if the specified pattern exists.

```
//Old
public void Baseline(string pattern, string measurePins, double waitTime = 0, double
measureWindow = 10 * ms, string config = "")
//New
public void Baseline(Pattern pattern, PinList pinList, double waitTime = 0, double
measureWindow = 10 * ms, string setup = "")
```

Impact: The test program verifies during validation whether the specified pattern exists.

Extend TestMethod Argument List

Adding `source` and `slope` parameters to the `Timing.Frequency.Baseline` TestMethod. Removing the default values from the `FrequencyCounter` Setup TestBlock.

Impact: User can specify the `source` and `slope` for the frequency read in the TestMethod params instead of in the TestBlock.

Implement Parametric Search

The following search methods have been implemented:

- `LinearFull` (*The measurements across the entire range are traversed without being evaluated during the linear search, after which the device input condition for which the output pin exceeds the threshold value is provided.*);
- `LinearStop` (*The measurements across the range are traversed with an evaluation performed at each iteration, and the search is stopped once the objective has been reached (on all sites). The device input condition for which the output pin exceeds the threshold value is then provided.*);

Implement Setup.Digital.ModifyXXX blocks

The following ModifyXXX methods have been implemented:

- `ModifyPins`: Selectively program or modify any digital instrument Pins parameter.
- `ModifyPinsLevels`: Selectively program or modify any digital instrument pins levels parameter.
- `ModifyPinsTiming`: Selectively program or modify any digital instrument pins timing parameter.

Rework PinSiteAddRange

Adding the capability to read more than one instrument in the methods implemented in **Acquire.Dc**, once the `.AddRange()` (from IG-XL 11) becomes available.

v0.6 - 2025-07-01

Breaking Changes

Adding the optional parameter `gate` to the setup functions.

Changes to the functions in the following way:

- The `connect` method will retain the option for `gate`, but it will be set to false by default instead of having `gate on` as the default.
- The `Force`, `ForceI`, `ForceV` and `SetForceAndMeter` methods will include an optional parameter to enable `gate`, which will, by default, switch to `gate on`.
- The `disconnect` method will keep the `gate` option and default to `gate off`.

The reasoning behind this is that it's safer to enable `gate` after the instrument is configured. In the current test methods, we enable `gate` in the `connect` method, but this approach involves connecting and powering the source with inherited settings, without knowing whether those settings are safe for the DUT.

Custom Action Use-Model Changed

Instead of having one use-model for custom settings, three different are introduced to target specific use-cases.

```
// previous use-model
Setting.Custom<T>(<string> key, T value, Action<T> setAction, Func<T[]> readFunc, T initialValue,
InitMode initMode, double settlingTime);

// new use-model
Setting.Custom<T>(T value, Action<T> setAction); // always executes
Setting.Custom<T>(<string> key, T value, Action<T> setAction, T initialValue, InitMode initMode);
// executes depending on cache
Setting.Custom<T>(<string> key, T value, Action<T> setAction, Func<T[]> readFunc, T initialValue,
InitMode initMode); // has also read capabilities
```

Impact: Users who have used the `Custom` Setting need to change their code to one of these three use-models.

Update & Enhance Test Blocks for `LinearFull`, `LinearStop` & `BinarySearch`

Functionality has been added to support searching for "closest-to-target" and overloads returning the search indices & output values for the input found. The method's signatures have been updated for

consistency.

Impact: Little impact as not widely (at all?) used so far.

Bug Fixes

Fixing the bug for measure current in MultiCondition

Voltage measurement was always selected.

Fixing the bug in MultiCondition

The issue regarding the for loop condition has been resolved, it now references the measured values. For displaying the forced value, a check was introduced to fix the bug when using uniform MultiCondition.

Solve the bug in Modify for PPMU

The syntax in method Modify for PPMU had an issue related to settings - if Mode and MeterMode were not set, an error was returned. This update ensures that the error is only triggered in specific cases where the required parameters are not provided.

Changes to the Force method

The issues that occurred for DCVS and DCVI are due to the `clampValue`. If the value does not belong to the range of clamp settings, errors will occur. Therefore, the call has been adjusted within the Force method so that the clamp value is passed as a range also.

Impact: On the user side, nothing changes.

Object initialization in IsValidated

Objects are not updated after validation if the user modifies the object (eg, pins) in the instance.

```
// Old
if (TheExec.Flow.IsValidated) {
    _pins ??= new Pins(pinList);
}
// New
if (TheExec.Flow.IsValidated) {
    _pins = new Pins(pinList);
}
```

Modification of the clamp setting for PPMU

The application of the clamp value in the case of PPMU was carried out in accordance with the forced current value when using method SetForceAndMeter.

Improvements

Add Domain to Pins type

Besides `Type` and `Feature`, a `Pins` object can now be queried for `Domain` information, like `Dc`, `Digital` or `Utility`.

```
Pins pins = new Pins("ana1, ana2, dig1");
if (!pins.ContainsDomain(InstrumentDomain.Dc)) {
    Console.WriteLine("no DC capable pins");
}
```

Impact: Simplifies test block and validation code to check if compatible pins have been provided.

Disconnecting/Connecting digital pins in TestMethods

Disconnecting/Connecting digital pins in TestMethods before connecting/disconnecting the DC part.
Modifying the documentation in accordance with TestMethods execution.

Pattern type as input variable

`Csra.Parametric.MultiCondition.PreconditionPattern` used a `string` instead of a `Pattern` type as an input variable.

Value-Based Comparison for Pin and Pins object

Two `Pin` objects are considered equal if their `Name` properties are equal. Two `Pins` collections are considered equal if they contain the same sequence of `Pin` objects (by name), in the same order.

v0.5 - 2025-06-02

Breaking Changes

Update outputMode for ForceV and Forcel methods

For the ForceV and Forcel methods, outputMode has been modified to use the Bool type instead of TLibOutputMode.

Explanation: Allowing settings such as "ForceCurrent" or "HighImpedance" in ForceV calls does not make sense. Now, switching to ForceVoltage is permitted, or the output mode change can be omitted if it was previously set, adjusting only the voltage. A similar behavior occurs in the Forcel method.

Enums Moving

The enums Kelvin, Measure, TLibOutputMode has been moved from Csra to Csra.V1 namespace.

```
// Old namespace reference
using Csra;

// New namespace reference
using Csra.V1;
```

Impact: Users need to update their code to reference the namespace Csra.V1 instead of Csra when using those enums.

Interface Moving

The interface ISetting has been moved from Csra to Csra.V1 namespace.

```
// Old namespace reference
using Csra;

// New namespace reference
using Csra.V1;
```

Impact: Users need to update their code to reference the namespace Csra.V1 instead of Csra when using the ISetting interface.

Support Pingroups

The SetupService now supports pingroups.

```

// Example of usage
Setup example = new("Normal");
example.Add(new
Setting.TheHdw.DcvI.Pins.FoldCurrentLimit.Behavior(tlDCVIFoldCurrentLimitBehavior.GateOff,
"pinName"));

Setting.TheHdw.DcvI.Pins.FoldCurrentLimit.Behavior.SetCache(tlDCVIFoldCurrentLimitBehavior.G
ateOff, "pinName");

```

Impact: The syntax for instantiating new settings has slightly changed. Instead of a 'List' of pins, now a single string as comma separated list is used. The string can contain single pins as well as pingroups. This syntax change also affects the 'SetCache()' method of each setting.

Bug Fixes

Adding Forced Value to Datalog

Parametric datalog did not show ForeValue - fixed.

Bug fix for current range application

The current value setting for DCVS depends on the current range value, which was omitted when using the Force method. In the ForceV and Forcel methods, it is included as an optional parameter. However, through the Force method, it cannot set the range value. Modifications have been made so that the range value is always applied without altering the method structure or its call.

Checking if PinList is null or empty in SingleCondition and Multi Condition.

The condition has been modified to check whether PinList is empty or null in these blocks.

Fixing the duplicate ForceVoltage error and missing ForceHiZ in switch

An error was identified in the switch logic used to separate ForceVoltage, ForceCurrent, and ForceHiZ, where ForceHiZ was not correctly included, and ForceVoltage appeared twice. The modification resolves this issue by correcting the switch branching, ensuring the proper distribution of values.

Remove StartState Assignments in .LevelsAndTiming.ApplyWithPinStates

Assignments to `StartState` have been removed for all pin groups (`initPinsHi`, `initPinsLo`, and `initPinsHiZ`). The method now only sets the `InitState`, aligning with the intended behavior described in the documentation.

Setup.LevelsAndTiming.ApplyWithPinStates now programs init states before ramping levels up or down

Impact: ApplyRepeatableSequences uses the init states to determine if levels are powered up or down. ApplyWithPinStates() now programs the init states before the levels are being ramped up or down. This change will only be visible for the parameter "levelRampSequence = True"

Improvements

Add TestBlocks for Search

Generic test blocks to allow building test methods for search & trim, using linear full, linear stop or binary search algorithms.

```
if (ShouldRunBody) {
    TheLib.Setup.Dc.ForceV(_inputPins, voltageFrom);
    TheLib.Setup.Dc.ForceHiZ(_outputPins);
    double increment = TheLib.Acquire.Search.LinearFullFromToSteps<double>(voltageFrom,
voltageTo, stepCount, ForceAndRead);
    _tripVoltage = TheLib.Execute.Search.LinearFullProcess(measurements, voltageFrom,
increment, 0, -999, m => m > outputThreshold);
}

void ForceAndRead(double inValue) {
    TheLib.Setup.Dc.ForceV(_inputPins, inValue);
    TheHdw.SettleWait(stepSettleTime);
    measurements.Add(TheLib.Acquire.Dc.Measure(_outputPins));
}
```

Impact: New functionality for integrated and performance optimized search and trim tests.

Implementation of the 'Modify()' test block

A new test block **TheLib.Setup.DC.Modify()** has been implemented, allowing the incremental modification of DC instrument properties.

```
private DcParameters _modifySettings;

[TestMethod, Steppable, CustomValidation]
public void Baseline(PinList pinList, ..., double clampHi, double clampLo, double
bandwidthSetting) {

    if (TheExec.Flow.IsValidating) {
```

```

        _modifySettings = new DcParameters() {
            ClampHiV = clampHi,
            ClampLoV = clampLo,
            BandwidthSet = bandwidthSetting
        };
    }

    if (ShouldRunBody) {
        TheLib.Setup.Dc.Modify(_pins, _modifySettings);
    }
}

```

Impact: The user has the ability to modify a series of properties in accordance with the necessary settings in the test block.

New Setting Support

The `SetupService` now supports `TheHdw.DcvI.Pins.BleederResistor.CurrentLoad` settings that can be used to configure the instruments.

```

// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.DcvI.Pins.BleederResistor.CurrentLoad(0.01, ["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");

```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.DcvI.Pins.BleederResistor.Mode` settings that can be used to configure the instruments.

```

// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.DcvI.Pins.BleederResistor.Mode(t1DCVIBleederResistor.Auto,
["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");

```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.DcvI.Pins.FoldCurrentLimit.Behavior` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new
Setting.TheHdw.DcvI.Pins.FoldCurrentLimit.Behavior(tlDCVIFoldCurrentLimitBehavior.GateOff,
["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.DcvI.Pins.FoldCurrentLimit.Timeout` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.DcvI.Pins.FoldCurrentLimit.Timeout(0.2, ["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.DcvI.Pins.NominalBandwidth` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.DcvI.Pins.NominalBandwidth(10, ["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.DcvS.Pins.CurrentLimit.Sink.FoldLimit.Level` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Dcvs.Pins.CurrentLimit.Sink.FoldLimit.Level(0.01,
["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.Dcvs.Pins.CurrentLimit.Sink.OverloadLimit.Level` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Dcvs.Pins.CurrentLimit.Sink.OverloadLimit.Level(0.01,
["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.Dcvs.Pins.CurrentLimit.Source.OverloadLimit.Level` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Dcvs.Pins.CurrentLimit.Source.OverloadLimit.Level(0.01,
["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.SetSettlingTimer` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.SetSettlingTimer(0.1));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.SettleWait` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.SettleWait(0.1));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

TestMethod Added

`Linear` and `LinearFull` `TestMethod` has been added.

Impact: Users can now use the `Search.Linear` or `Search.LinearFull` at the `TestInstance` sheet.

v0.4 - 2025-05-09

Breaking Changes

Class Renaming

The class `SetupX` has been renamed to `Setup`.

```
// Old class reference
SetupX setup = new SetupX("xx");

// New class reference
Setup setup = new Setup("xx");
```

Impact: Users need to update their code to reference the new class name `Setup` instead of `SetupX`.

Class Renaming

The class `StpHdw` has been renamed to `Setting.TheHdw`.

```
// Old class reference
SetupX normal = new("Normal");
normal.Add(new StpHdw.Utility.Pins.State(tlUtilBitState.Off, ["K2", "K3", "K5", "K6"]));

// New class reference
Setup normal = new("Normal");
normal.Add(new Setting.TheHdw.Utility.Pins.State(tlUtilBitState.Off, ["K2", "K3",
"K5", "K6"]));
```

Impact: Users need to update their code to reference the new class name `Setting.TheHdw` instead of `StpHdw`.

Enum Renaming

The enums `Feature` and `Type` of the `Pins` object has been moved outside of the `Pins` object and renamed to `InstrumentFeature` and `InstrumentType`.

```
// Old enum reference
var feature = Pins.Feature.Ppmu;
var type = Pins.Type.UVS256;

// New enum reference
```

```
var feature = InstrumentFeature.Ppmu;  
var type = InstrumentType.UVS256;
```

Impact: Users need to update their code to reference the new enum names `InstrumentFeature` and `InstrumentType` instead of `Feature` and `Type`.

Method Renaming

Services that previously had a `Clear` method have renamed it to `Reset`. Services that did not have a `Clear` method now have a `Reset` method. The `Reset` method will reset all states of the services.

```
// Old method reference  
service.Clear();  
  
// New method reference  
service.Reset();
```

Impact: Users need to update their code to use the `Reset` method for all services.

Method Renaming

The method `ApplyLevelsTiming` has been split into two methods named `LevelsAndTiming.Apply` and `LevelsAndTiming.ApplyWithPinStates`.

```
// Old method reference  
TheLib.ApplyLevelsTiming(xxx);  
  
// New method reference  
TheLib.LevelsAndTiming.ApplyWithPinStates(xxx);  
TheLib.LevelsAndTiming.Apply(xxx);
```

Impact: Users need to update their code to reference the new method name `ApplyWithPinStates` or `Apply` instead of `ApplyLevelsTiming`.

Namespace Renaming

The namespace `ReferenceArchitecture` has been renamed to `Csra`.

```
// Old namespace reference  
using static ReferenceArchitecture.V1.Api;  
  
// New namespace reference  
using static Csra.V1.Api;
```

Impact: Users need to update their code to reference the new namespace `Csra` instead of `ReferenceArchitecture`. The IG-XL instance sheet needs to be updated as well to use `Csra`.

Parameter Update

The method `Connect` now has a parameter `gateOn` with a default value of `true` and `Disconnect` has a parameter `gateOff` with a default value of `true` instead of a nullable parameter.

```
// Old method signatures
TheLib.Setup.Dc.Connect(Pins pins, bool? gateOn = null);
TheLib.Setup.Dc.Disconnect(Pins pins, Boolean? gateOn = null);

// New method signatures with default parameter value
TheLib.Setup.Dc.Connect(Pins pins, bool gateOn = true);
TheLib.Setup.Dc.Disconnect(Pins pins, bool gateOff = true);
```

Impact: Users need to update their code to handle the `gateOff` and `gateOn` parameter as a non-nullable boolean, which defaults to true.

Project Renaming

The C# project has been renamed from ADU to Demo.

```
// Old project reference
using ADU;

// New project reference
using Demo;
```

Impact: Users need to update their project references and namespaces from ADU to Demo to ensure compatibility with the new project structure.

Bug Fixes

DCVI Instrument Voltage Clamp Setting

Fixed an issue where the clamp setting for a DCVI instrument was missing. The fix ensures that the clamp setting is now correctly applied.

Impact: Users can now correctly set the clamp for the instrument, ensuring proper functionality and preventing potential errors.

DCVS Instrument Sink Limit restrictions

Fixed an issue where it was possible to set the DCVS instrument outside of its sink limits. The fix ensures that values are now clamped at the maximum limits.

Impact: Users can no longer set the DCVS instrument to values outside of its defined limits, ensuring proper functionality and preventing potential errors.

Initialization Timing in DriverMode

Fixed an issue where the value in the setting `Setting.TheHdw.Digital.Pins.Levels.DriverMode` was incorrectly initialized at `Creation` time instead of during a `OnProgramStarted` event. This change ensures the value now accurately reflects the hardware behavior.

Impact: Users do not need to take any action. The fix ensures the value initialization now correctly aligns with the hardware behavior, eliminating potential timing issues.

ReferenceArchitecture Properties

Fixed an issue where the ReferenceArchitecture has copied the IG-XL dll locally instead of using the `IGXLROOT` environment variable, as Teradyne suggests.

Impact: Users do not need to set the property manually to false.

Improvements

Default Value Change

The default value of the `RespectSettlingTimeDefault` property of the `SetupService` has been changed from `true` to `false`. This change eliminates potential wait time between calls, reducing timing issues.

Impact: Users need to be aware of the change in default behavior, which now avoids potential timing issues by not adding wait time between calls.

Method Addition

A new method `SetForceAndMeter` has been added to consolidate the functionality of `SetForce` and `SetMeter`.

```
// Old method calls
TheLib.Setup.Dc.ForceI(Pins pins, double forceCurrent, double? clampVoltage = null);
// or
TheLib.Setup.Dc.ForceV(Pins pins, double forceVoltage, double? clampCurrent = null);
TheLib.Setup.Dc.SetMeter(Pins pins, Measure meterMode, double rangeValue, double?
filterValue = null, int? hardwareAverage = null, double? outputRangeValue = null);

// New method call
```

```
TheLib.Setup.Dc.SetForceAndMeter(Pins pins, TLibOutputMode mode, double forceValue, double forceRange, double clampValue, Measure meterMode, double measureRange);
```

Impact: Users can now use the `SetForceAndMeter` method for convenience, while the existing `SetForce` and `SetMeter` methods remain available.

Mock Injection Support

Added functionality to enable mock injection, allowing users to write unit tests for their code using C#RA. This feature supports mocking `TheLib`, `AlertService`, `SetupService`, `StorageService`.

```
// bla
SetupCsraMoq(ILib lib = null, IAlertService alert = null, IStorageService storage = null,
ISetupService setup = null)

// Example of mock injection
var mockTheLib = new Mock<ITheLib>();
SetupCsraMoq(mockTheLib.Object);

TheLib.Setup.Dc.Connect(pins, true);
mockTheLib.Verify(lib => lib.Setup.Dc.Connect(pins, true), Times.Once);
```

Impact: Users can now easily create mock objects for unit testing, enhancing their ability to write comprehensive unit tests and improving test coverage and reliability of their code.

New Setting Support

The `SetupService` now supports `TheHdw.Dcv.Pins.ComplianceRange_Negative` and `TheHdw.Dcv.Pins.ComplianceRange_Positive` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Dcv.Pins.ComplianceRange_Negative(-2, ["pinName"]));
example.Add(new Setting.TheHdw.Dcv.Pins.ComplianceRange_Positive(1, ["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.Dcv.Pins.BleederResistor` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Dcvs.Pins.BleederResistor(tlDCVSOnOffAuto.On, ["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.Digital.Pins.Levels.DriverMode` settings that can be used to configure the device.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Digital.Pins.Levels.DriverMode(tlDriverMode.Hiz,
["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.Digital.Pins.Levels.Value_Ioh` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Digital.Pins.Levels.Value_Ioh(0.3, ["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.Digital.Pins.Levels.Value_Iol` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Digital.Pins.Levels.Value_Iol(0.3, ["pinName"]));
```

```
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.Digital.Pins.Levels.Value_Vch` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Digital.Pins.Levels.Value_Vch(0.3, ["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.Digital.Pins.Levels.Value_Vcl` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Digital.Pins.Levels.Value_Vcl(0.3, ["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.Digital.Pins.Levels.Value_Vih` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Digital.Pins.Levels.Value_Vih(0.3, ["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.Digital.Pins.Levels.Value_Vil` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Digital.Pins.Levels.Value_Vil(0.3, ["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.Digital.Pins.Levels.Value_Voh` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Digital.Pins.Levels.Value_Voh(0.3, ["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.Digital.Pins.Levels.Value_Vol` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Digital.Pins.Levels.Value_Vol(0.3, ["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

New Setting Support

The `SetupService` now supports `TheHdw.Digital.Pins.Levels.Value_Vt` settings that can be used to configure the instruments.

```
// Example of using the new setting
Setup example = new("Normal");
example.Add(new Setting.TheHdw.Digital.Pins.Levels.Value_Vt(0.3, ["pinName"]));
SetupService.Add(example);
SetupService.Apply("Normal");
```

Impact: Users can now utilize the new setting to customize the `SetupService` according to their needs.

SetupService Customer Setting

The `SetupService` now supports customers in setting up their own settings.

Impact: Users can now customize their settings using the `SetupService`, providing greater flexibility and control.

String Handling

The apply-method of the `SetupService` now supports a comma-separated list as a string, in addition to a normal string.

```
// Old method calls
SetupService.Apply("setup1");
SetupService.Apply("setup2");

// New method call
SetupService.Apply("setup1, setup2"); // valid
```

Impacts: Users can now pass a comma-separated list to the method, allowing for more flexible data handling.

Namespace Csra

Namespaces

[Csra.Interfaces](#)

[Csra.Services](#)

[Csra.Settings](#)

[Csra.TheLib](#)

Classes

[BiDictionary< TKey, TValue >](#)

Represents a bidirectional dictionary that allows lookups in both directions.

[CSRATransactionConfig](#)

[CSRATransactionConfig.Port](#)

[CoreInstanceTestResult](#)

Class to store the test result of a core instance, part of [ScanNetworkPatternResults](#) of a ScanNetwork pattern(set).

[DcParameters](#)

[DigitalPinsLevelsParameters](#)

[DigitalPinsParameters](#)

[DigitalPinsTimingParameters](#)

[ExtensionMethods](#)

[IclInstanceInfo](#)

Class to store the attributes of an icl instance, part of [ScanNetworkPatternInfo](#) for a ScanNetwork pattern(set).

[IclInstanceTestResult](#)

Class to store the metadata and test result of an icl instance, part of [ScanNetworkPatternResults](#) of a ScanNetwork pattern(set).

[MethodHandleTargetAttribute](#)

Indicates that the target method is intended to be referenced via a method handle, such as a delegate.

[MethodHandle<T>](#)

Provides a flexible and type-safe mechanism for dynamically resolving and invoking delegates based on the execution context (e.g., debug vs. production).

[PatternInfo](#)

Class to store information about a pattern.

[Pins](#)

Pins class - a collection of Pin objects.

[Pins.Pin](#)

Pin class - individual hardware pins with features.

[ScanNetworkPatternInfo](#)

Class to store ScanNetwork information about a ScanNetwork pattern(set).

[ScanNetworkPatternResults](#)

Class to store the per core/icl instance test results of a ScanNetwork pattern(set).

[Setup](#)

Setup - a named collection of settings.

[SsnCsvFile](#)

Class to parse information from an ssn.csv file. Only for use in constructor method of [ScanNetwork PatternInfo](#).

Enums

[AlertOutputTarget](#)

The available output targets for Alert Service messages.

[InitMode](#)

Events in the lifetime of a Setup that will cause a reset of the hardware state.

[InstrumentDomain](#)

Functional domain available by the instrument.

[InstrumentFeature](#)

Logical features provided through the instrument driver.

[InstrumentType](#)

Physical instrument types.

[Kelvin](#)

Measure

ScanNetworkDatalogOption

Specifies the options for logging ScanNetwork test results from object [ScanNetworkPatternResults](#).

TLibDiffLv\ValType

TLibOutputMode

TransactionType

Transaction type to use with the TransactionService.

Namespace Csra.Interfaces

Interfaces

[IAlertService](#)

[IBehaviorService](#)

[ILib](#)

The interface for the EntryPoint.

[ILib.IAcquire](#)

The interface for the Acquire branch.

[ILib.IAcquire.IDc](#)

The interface for the Dc branch.

[ILib.IAcquire.IDigital](#)

The interface for the Digital branch.

[ILib.IAcquire.IScanNetwork](#)

The interface for the ScanNetwork branch.

[ILib.IAcquire.ISearch](#)

The interface for the Search branch.

[ILib.IDatalog](#)

The interface for the Datalog branch.

[ILib.IExecute](#)

The interface for the Execute branch.

[ILib.IExecute.IDigital](#)

The interface for the Digital branch.

[ILib.IExecute.IScanNetwork](#)

The interface for the ScanNetwork branch.

[ILib.IExecute.ISearch](#)

The interface for the Search branch.

[ILib.ISetup](#)

The interface for the Setup branch.

[ILib.ISetup.IDc](#)

The interface for the Dc branch.

[ILib.ISetup.IDigital](#)

The interface for the Digital branch.

[ILib.ISetup.ILevelsAndTiming](#)

The interface for the LevelsAndTiming branch.

[ILib.IValidate](#)

[IService](#)

[ISetting](#)

[ISetupService](#)

[IStorageService](#)

[ITransactionConfig](#)

[ITransactionService](#)

Interface IAlertService

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

```
public interface IAlertService : IService
```

Inherited Members

[IService.Reset\(\)](#)

Properties

ErrorTarget

Reads or sets the output target for Error Alerts. Defaults to OutputWindow and Datalog, which cannot be disabled.

```
AlertOutputTarget ErrorTarget { get; set; }
```

Property Value

[AlertOutputTarget](#)

InfoTarget

Reads or sets the output target for Info Alerts. Defaults to OutputWindow, which cannot be disabled.

```
AlertOutputTarget InfoTarget { get; set; }
```

Property Value

[AlertOutputTarget](#)

LogTarget

Reads or sets the output target for Log Alerts. Defaults to OutputWindow, which cannot be disabled.

```
AlertOutputTarget LogTarget { get; set; }
```

Property Value

[AlertOutputTarget](#)

OutputFile

Reads or sets the file path for the output file.

```
string OutputFile { get; set; }
```

Property Value

[string](#)

TimeStamp

Reads or sets whether a time stamp is added to Info / Warning and Error Alerts.

```
bool TimeStamp { get; set; }
```

Property Value

[bool](#)

WarningTarget

Reads or sets the output target for Warning Alerts. Defaults to OutputWindow and Datalog, which cannot be disabled.

```
AlertOutputTarget WarningTarget { get; set; }
```

Property Value

[AlertOutputTarget](#)

Methods

Error(string, int, string)

Sends an Error Alert message to the selected output target(s). At test program runtime, this raises an exception to the IG-XL error handler. If called during validation, it fails validation and flags the error appropriately. Use this for non-recoverable conditions that require immediate and safe termination.

Note: The compiler does not recognize that this method never returns, so you may need extra checks to satisfy .NET Framework requirements.

```
void Error(string error, int validationArgumentIndex = 0, string doNotSpecify = "")
```

Parameters

error [string](#)

The Error Alert message.

validationArgumentIndex [int](#)

Optional. The offending test instance argument index (one-based), if applicable when used in validation.

doNotSpecify [string](#)

DO NOT SPECIFY - the name of the calling method is automatically inserted by the compiler.

Error<TException>(string, int, string)

Sends an Error Alert message to the selected output target(s). At test program runtime, this raises a user-selectable exception to the IG-XL error handler. If called during validation, it fails validation and flags the error appropriately. Use this for non-recoverable conditions that require immediate and safe termination.

Note: The compiler does not recognize that this method never returns, so you may need extra checks to satisfy .NET Framework requirements.

```
void Error<TException>(string error, int validationArgumentIndex = 0, string doNotSpecify = "") where TException : Exception
```

Parameters

error [string](#)

The Error Alert message.

validationArgumentIndex [int](#)

Optional. The offending test instance argument index (one-based), if applicable when used in validation.

doNotSpecify [string](#)

DO NOT SPECIFY - the name of the calling method is automatically inserted by the compiler.

Type Parameters

TException

The type of the exception to be thrown.

Info(string, string)

Sends an Info Alert message to the selected output target(s). Use for positive / neutral information relevant to the user.

```
void Info(string info, string doNotSpecify = "")
```

Parameters

info [string](#)

The Info Alert message.

doNotSpecify [string](#)

DO NOT SPECIFY - the name of the calling method is automatically inserted by the compiler.

Log(string, byte, byte, byte, bool)

Sends a Log Alert message to the selected output target(s).

```
void Log(string message, byte red, byte green, byte blue, bool bold = false)
```

Parameters

message [string](#)

The message string to be logged.

red [byte](#)

The red component of a RGB color.

green [byte](#)

The green component of a RGB color.

blue [byte](#)

The blue component of a RGB color.

bold [bool](#)

Optional. Whether bold font is used (OutputWindow only, ignored elsewhere).

Log(string, ColorConstants, bool)

Sends a Log Alert message to the selected output target(s).

```
void Log(string message, ColorConstants color = ColorConstants.Black, bool bold = false)
```

Parameters

message [string](#)

The message string to be logged.

color ColorConstants

Optional. The color to be used (OutputWindow only, ignored elsewhere).

bold [bool](#)

Optional. Whether bold font is used (OutputWindow only, ignored elsewhere).

Warning(string, string)

Sends a Warning Alert message to the selected output target(s). Use for recoverable issues that may require attention.

```
void Warning(string warning, string doNotSpecify = "")
```

Parameters

warning [string](#)

The Warning Alert message.

doNotSpecify [string](#)

DO NOT SPECIFY - the name of the calling method is automatically inserted by the compiler.

Interface IBehaviorService

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

```
public interface IBehaviorService : IService
```

Inherited Members

[IService.Reset\(\)](#)

Properties

Features

Gets a collection containing all defined features.

```
IEnumerable<string> Features { get; }
```

Property Value

[IEnumerable<string>](#)

FilePath

Gets/Sets the import / export file path.

```
string FilePath { get; set; }
```

Property Value

[string](#)

Methods

Export(string)

Writes all features to the specified file, or a previously defined [FilePath](#) if empty. Updates the [FilePath](#) setting.

```
void Export(string filePath = "")
```

Parameters

filePath [string](#)

Optional. The (relative or absolute) file path.

GetFeature<T>(string)

Reads a feature's value. Type must match the original definition, an exception is thrown otherwise.

```
T GetFeature<T>(string feature)
```

Parameters

feature [string](#)

The feature name.

Returns

T

The feature's value.

Type Parameters

T

The feature value's type.

Import(string)

Reads the specified file, or a previously defined [FilePath](#) if empty. Incrementally updates features with new values. Call [Services.Behavior.Reset\(\)](#) to clear all data first. Updates the [FilePath](#) setting.

```
void Import(string filePath = "")
```

Parameters

filePath [string](#)

Optional. The (relative or absolute) file path.

SetFeature<T>(string, T)

Defines a feature's value. Creates a new entry, or updates an existing one if it already exists.

```
void SetFeature<T>(string feature, T value)
```

Parameters

feature [string](#)

The feature name.

value [T](#)

The feature's new value.

Type Parameters

T

The feature value's type.

Interface ILib

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the EntryPoint.

```
public interface ILib
```

Properties

Acquire

The accessor for the Acquire branch.

```
ILib.IAcquire Acquire { get; }
```

Property Value

[ILib.IAcquire](#)

Datalog

The accessor for the Datalog branch.

```
ILib.IDatalog Datalog { get; }
```

Property Value

[ILib.IDatalog](#)

Execute

The accessor for the Execute branch.

```
ILib.IExecute Execute { get; }
```

Property Value

[ILib.IExecute](#)

Setup

The accessor for the Setup branch.

```
ILib.ISetup Setup { get; }
```

Property Value

[ILib.ISetup](#)

Validate

The accessor for the Validate branch.

```
ILib.IValidate Validate { get; }
```

Property Value

[ILib.IValidate](#)

Interface ILib.IAcquire

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the Acquire branch.

```
public interface ILib.IAcquire
```

Properties

Dc

The accessor for the Dc branch.

```
ILib.IAcquire.IDc Dc { get; }
```

Property Value

[ILib.IAcquire.IDc](#)

Digital

The accessor for the Digital branch.

```
ILib.IAcquire.IDigital Digital { get; }
```

Property Value

[ILib.IAcquire.IDigital](#)

ScanNetwork

The accessor for the ScanNetwork branch.

```
ILib.IAcquire.IScanNetwork ScanNetwork { get; }
```

Property Value

[ILib.IAcquire.IScanNetwork](#)

Search

The accessor for the Search branch.

```
ILib.IAcquire.ISearch Search { get; }
```

Property Value

[ILib.IAcquire.ISearch](#)

Interface ILib.IAcquire.IDc

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the Dc branch.

```
public interface ILib.IAcquire.IDc
```

Methods

Measure(Pins, int, double?, Measure?)

Performs multiple measurements for the set of pins provided.

```
PinSite<double> Measure(Pins pins, int sampleSize, double? sampleRate = null, Measure?  
meterMode = null)
```

Parameters

pins [Pins](#)

The pins to measure.

sampleSize [int](#)

The number of samples.

sampleRate [double](#)?

Optional. The sampling rate.

meterMode [Measure](#)?

Optional. Set the mode to measure Voltage or Current.

Returns

[PinSite<double>](#)

Returns an average value.

Exceptions

[Exception ↗](#)

Appears when pinList contains different types of pins - temporary limitation in functionality.

Measure(Pins, Measure?)

Performs a single measurement for the set of pins provided.

```
PinSite<double> Measure(Pins pins, Measure? meterMode = null)
```

Parameters

pins [Pins](#)

The pins to measure.

meterMode [Measure?](#)

Optional. Set the mode to measure Voltage or Current.

Returns

```
PinSite<double>
```

Returns a value.

Exceptions

[Exception ↗](#)

Appears when pinList contains different types of pins - temporary limitation in functionality.

Measure(Pins[], int[], double[], Measure[])

Performs multiple measurements for the set of each element in pinGroups.

```
PinSite<double> Measure(Pins[] pinGroups, int[] sampleSizes, double[] sampleRates = null,  
Measure[] meterModes = null)
```

Parameters

pinGroups [Pins\[\]](#)

Array of pin or pin groups.

sampleSizes [int\[\]](#)

Array of number of samples.

sampleRates [double\[\]](#)

Optional. Array of sampling rate.

meterModes [Measure\[\]](#)

Optional. Array of settings measurements mode voltage and current.

Returns

[PinSite<double>](#)

Returns a set of measurements.

Exceptions

[Exception](#)

Appears when an element of pinGroups contains different types of pins - temporary limitation in functionality.

MeasureSamples(Pins, int, double?, Measure?)

Performs multiple measurements for the set of pins provided.

```
PinSite<Samples<double>> MeasureSamples(Pins pins, int sampleSize, double? sampleRate =  
null, Measure? meterMode = null)
```

Parameters

`pins` [Pins](#)

The pins to measure.

`sampleSize` [int](#)

The number of samples.

`sampleRate` [double](#)?

Optional. The sampling rate.

`meterMode` [Measure](#)?

Optional. Set the mode to measure Voltage or Current.

Returns

`PinSite<Samples<double>>`

Returns a set of measurements.

Exceptions

[Exception](#)

Appears when pinList contains different types of pins - temporary limitation in functionality.

MeasureSamples(Pins[], int[], double[], Measure[])

Performs multiple measurements for the set of each element in pinGroups.

```
PinSite<Samples<double>> MeasureSamples(Pins[] pinGroups, int[] sampleSizes, double[] sampleRates = null, Measure[] meterModes = null)
```

Parameters

`pinGroups` [Pins](#)[]

Array of pin or pin groups.

`sampleSizes` [int](#)[]

Array of number of samples.

`sampleRates` [double](#)[]

Optional. Array of sampling rate.

`meterModes` [Measure](#)[]

Optional. Array of settings measurements mode voltage and current.

Returns

`PinSite<Samplesdouble>>`

Returns a set of measurements.

Exceptions

[Exception](#)

Appears when an element of pinGroups contains different types of pins - temporary limitation in functionality.

ReadCaptured(Pins, string)

Allows configuration and control of capture parameters for the specified pins.

`PinSite<double> ReadCaptured(Pins pins, string signalName)`

Parameters

`pins` [Pins](#)

List of pin or pin group names.

`signalName` [string](#)

The name of the read signal.

Returns

PinSite<[double](#)>

Returns a value.

Exceptions

[Exception](#)

Appears when pinList contains different types of pins - temporary limitation in functionality.

ReadCapturedSamples(Pins, string)

Allows configuration and control of capture parameters for the specified pins.

PinSite<Samples<[double](#)>> ReadCapturedSamples(Pins pins, [string](#) signalName)

Parameters

[pins](#) [Pins](#)

List of pin or pin group names.

[signalName](#) [string](#)

The name of the read signal.

Returns

PinSite<Samples<[double](#)>>

Returns a set of measurements.

Exceptions

[Exception](#)

Appears when pinList contains different types of pins - temporary limitation in functionality.

ReadMeasured(Pins, int, double?)

Performs multiple readings of measurements, depending on the sample size parameter, for the set of pins provided.

```
PinSite<double> ReadMeasured(Pins pins, int sampleSize, double? sampleRate = null)
```

Parameters

pins [Pins](#)

The pins to read measurements.

sampleSize [int](#)

The number of samples.

sampleRate [double](#)?

Optional. The sampling rate.

Returns

[PinSite<double>](#)

Returns an average value.

Exceptions

[Exception](#)

Appears when pinList contains different types of pins - temporary limitation in functionality.

ReadMeasuredSamples(Pins, int, double?)

Performs multiple readings of measurements, depending on the sample size parameter, for the set of pins provided.

```
PinSite<Samples<double>> ReadMeasuredSamples(Pins pins, int sampleSize, double? sampleRate = null)
```

Parameters

`pins` [Pins](#)

The pins to read measurements.

`sampleSize` [int](#)

The number of samples.

`sampleRate` [double](#)?

Optional. The sampling rate.

Returns

`PinSite<Samples<double>>`

Returns a set of measurements.

Exceptions

[Exception](#)

Appears when pinList contains different types of pins - temporary limitation in functionality.

Strobe(Pins)

Performs a single measurement (strobe) on the according instrument, to read back the value later.

`void Strobe(Pins pins)`

Parameters

`pins` [Pins](#)

The pins to be measured.

StrobeSamples(Pins, int, double?)

Performs multiple measurements (strokes) on the according instrument, to read back the value later.

```
void StrobeSamples(Pins pins, int sampleSize, double? sampleRate = null)
```

Parameters

pins [Pins](#)

The pins to be measured.

sampleSize [int](#)?

Number of measurements (Strobes).Ignored for PPMU.

sampleRate [double](#)?

Optional. The sampling rate. Default is the current hardware setting. Ignored for PPMU.

Interface ILib.IAcquire.IDigital

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the Digital branch.

```
public interface ILib.IAcquire.IDigital
```

Methods

MeasureFrequency(Pins)

Measures and returns the frequency configured by Setup.Digital.FrequencyCounter.

```
PinSite<double> MeasureFrequency(Pins pins)
```

Parameters

pins [Pins](#)

List of pin or pin group names.

Returns

```
PinSite<double>
```

The measured frequency for each pin in Hz.

PatternResults()

```
Site<bool> PatternResults()
```

Returns

```
Site<bool>
```

Read(Pins, int, int)

Reads captured pin data from HRAM and returns raw results as IPinListData

```
PinSite<Samples<int>> Read(Pins pins, int startIndex = 0, int cycle = 0)
```

Parameters

pins [Pins](#)

Pin names, must contain digital pins

startIndex [int](#)

Optional. Index to start capture

cycle [int](#)

Optional. Cycle of data to capture for pins in 2x or 4x modes

Returns

[PinSite<Samples<int>>](#)

Raw captured pin data

Exceptions

[ArgumentException](#)

ReadWords(Pins, int, int, int, tlBitOrder)

Reads pin data from specified cycles in HRAM for the specified pin, groups the data into words of a specified size, and populates these words into an array of SiteLong objects

```
PinSite<Samples<int>> ReadWords(Pins pins, int startIndex, int length, int wordSize,  
tlBitOrder bitOrder)
```

Parameters

pins [Pins](#)

Pin names, digital pins required

startIndex [int](#)

Index to start data processing

length [int](#)

Number of bits to process

wordSize [int](#)

Number of bits in each word

bitOrder [tlBitOrder](#)

Order of bits, either msbForst or lsbFirst

Returns

`PinSite<Samples<int>>`

Word values in array of `ISiteLong`

Exceptions

[ArgumentException](#)

Interface ILib.IAcquire.IScanNetwork

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the ScanNetwork branch.

```
public interface ILib.IAcquire.IScanNetwork
```

Methods

PatternResults(ScanNetworkPatternInfo)

Retrieve the per core/icl instance results of the specified ScanNetwork pattern object from its latest execution (TheLib.Execute.ScanNetwork.RunPattern(ScanNetworkPatternObject)).

```
ScanNetworkPatternResults PatternResults(ScanNetworkPatternInfo scanNetworkPattern)
```

Parameters

`scanNetworkPattern` [ScanNetworkPatternInfo](#)

The [ScanNetworkPatternInfo](#) Object that is associated with the ScanNetwork pattern(set).

Returns

[ScanNetworkPatternResults](#)

[ScanNetworkPatternResults](#) object that contains per core/icl instance results

Interface ILib.IAcquire.ISearch

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the Search branch.

```
public interface ILib.IAcquire.ISearch
```

Methods

BinarySearch<Tout>(double, double, double, bool, Func<Site<double>, Site<Tout>>, Func<Tout, bool>, double)

Performs a floating point binary search between `inFrom` and `inTo` by executing `oneMeasurement` at each step. Determines the input resulting in an output closest to the trip criteria's inflection point. The number of steps executed equals $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the minimum delta is reached, the reported result lies within `inMinDelta` from the ideal result, matching criteria's direction. If no transition was found, the method returns `inNotFoundResult`.

```
Site<double> BinarySearch<Tout>(double inFrom, double inTo, double inMinDelta, bool
invertingLogic, Func<Site<double>, Site<Tout>> oneMeasurement, Func<Tout, bool>
outTripCriteria, double inNotFoundResult)
```

Parameters

`inFrom` [double](#)

The lower boundary of the search range. Must be less than `inTo`.

`inTo` [double](#)

The upper boundary of the search range. Must be greater than `inFrom`

`inMinDelta` [double](#)

The minimum allowable difference between successive input values, used to determine when the search should stop.

`invertingLogic` `bool`

Determines whether the logic is inverted. `false` means higher input values increase the likelihood of meeting the trip criteria. `true` means lower input values do.

`oneMeasurement` `Func<Site<double>, Site<Tout>>`

The action to execute for every measurement.

`outTripCriteria` `Func<Tout, bool>`

A delegate indicating the output meets the condition required for the input value searched.

`inNotFoundResult` `double`

The return value for the case when the trip criteria was never found.

Returns

`Site<double>`

The input value resulting in an output fulfilling the trip criteria and be closest to it's inflection point.

The worst case deviation from the exact input is (`inMinDelta`.

Type Parameters

`Tout`

The type of the device's output.

`BinarySearch<Tout>(double, double, double, bool, Func<Site<double>, Site<Tout>>, Func<Tout, bool>, double, out Site<Tout>)`

Performs a floating point binary search between `inFrom` and `inTo` by executing `oneMeasurement` at each step. Determines the input resulting in an output closest to the trip criteria's inflection point. The number of steps executed equals $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the minimum delta is reached, the reported result lies within `inMinDelta` from the ideal result, matching criteria's direction. Additionally provides the output value of the input step found. If no transition was found, the method returns `inNotFoundResult`.

```
Site<double> BinarySearch<Tout>(double inFrom, double inTo, double inMinDelta, bool  
invertingLogic, Func<Site<double>, Site<Tout>> oneMeasurement, Func<Tout, bool>  
outTripCriteria, double inNotFoundResult, out Site<Tout> outResult)
```

Parameters

inFrom [double](#)

The lower boundary of the search range. Must be less than **inTo**.

inTo [double](#)

The upper boundary of the search range. Must be greater than **inFrom**

inMinDelta [double](#)

The minimum allowable difference between successive input values, used to determine when the search should stop.

invertingLogic [bool](#)

Determines whether the logic is inverted. **false** means higher input values increase the likelihood of meeting the trip criteria. **true** means lower input values do.

oneMeasurement [Func](#)<Site<double>, Site<Tout>>

The action to execute for every measurement.

outTripCriteria [Func](#)<Tout, [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

inNotFoundResult [double](#)

The return value for the case when the trip criteria was never found.

outResult Site<Tout>

Output - contains the output value of the input found.

Returns

Site<[double](#)>

The input value resulting in an output fulfilling the trip criteria and be closest to it's inflection point.

The worst case deviation from the exact input is ([inMinDelta](#).

Type Parameters

Tout

The type of the device's output.

BinarySearch<Tout>(double, double, double, bool, Func<Site<double>, Site<Tout>>, Tout)

Performs a floating point binary search between [inFrom](#) and [inTo](#) by executing [oneMeasurement](#) at each step. Determines the input resulting in an output closest to the (numeric) target. The number of steps executed equals $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the minimum delta is reached, the reported result lies within +/- [inMinDelta](#) from the ideal result.

```
Site<double> BinarySearch<Tout>(double inFrom, double inTo, double inMinDelta, bool  
invertingLogic, Func<Site<double>, Site<Tout>> oneMeasurement, Tout outTarget)
```

Parameters

[inFrom](#) [double](#)

The lower boundary of the search range. Must be less than [inTo](#).

[inTo](#) [double](#)

The upper boundary of the search range. Must be greater than [inFrom](#)

[inMinDelta](#) [double](#)

The minimum allowable difference between successive input values, used to determine when the search should stop. Must be >0.

[invertingLogic](#) [bool](#)

A flag indicating whether the output is inverted, meaning increasing output values are a result of decreasing input values.

[oneMeasurement](#) [Func](#)<[Site<double>](#), Site<Tout>>

The action to execute for every measurement.

outTarget Tout

The (numeric) target output value for which the corresponding input condition is searched.

Returns

Site<[double](#)>

The input value resulting in an output closest to the target. The worst case delta to the exact input is $+/- (\text{inMinDelta} / 2)$ if it can be reached given the search range.

Type Parameters

Tout

The type of the device's output.

BinarySearch<Tout>(double, double, double, bool, Func<Site<double>, Site<Tout>>, Tout, out Site<Tout>)

Performs a floating point binary search between `inFrom` and `inTo` by executing `oneMeasurement` at each step. Determines the input resulting in an output closest to the (numeric) target. The number of steps executed equals $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the minimum delta is reached, the reported result lies within $+/- \text{inMinDelta}$ from the ideal result. Additionally provides the output value for the input step found.

```
Site<double> BinarySearch<Tout>(double inFrom, double inTo, double inMinDelta, bool
invertingLogic, Func<Site<double>, Site<Tout>> oneMeasurement, Tout outTarget, out
Site<Tout> outResult)
```

Parameters

`inFrom` [double](#)

The lower boundary of the search range. Must be less than `inTo`.

`inTo` [double](#)

The upper boundary of the search range. Must be greater than `inFrom`

inMinDelta [double](#)

The minimum allowable difference between successive input values, used to determine when the search should stop.

invertingLogic [bool](#)

A flag indicating whether the output is inverted, meaning increasing output values are a result of decreasing input values.

oneMeasurement [Func](#) <Site<[double](#)>, Site<Tout>>

The action to execute for every measurement.

outTarget Tout

The (numeric) target output value for which the corresponding input condition is searched.

outResult Site<Tout>

Output - contains the output value for the input found.

Returns

Site<[double](#)>

The input value resulting in an output closest to the target. The worst case delta to the exact input is +/- (**inMinDelta** / 2) if it can be reached given the search range.

Type Parameters

Tout

The type of the device's output.

BinarySearch<Tout>(int, int, int, bool, Func<Site<int>, Site<Tout>>, Func<Tout, bool>, int)

Performs an integer binary search between **inFrom** and **inTo** by executing **oneMeasurement** at each step. Determines the input resulting in an output closest to the trip criteria's inflection point. The number of steps executed does not exceed $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the input value closest to the transition point (within the specified **inMinDelta**

resolution), but matching the criteria is reached. If no transition was found, the method returns `inNotFoundResult`.

```
Site<int> BinarySearch<Tout>(int inFrom, int inTo, int inMinDelta, bool invertingLogic,  
Func<Site<int>, Site<Tout>> oneMeasurement, Func<Tout, bool> outTripCriteria,  
int inNotFoundResult)
```

Parameters

`inFrom` [int](#)

The lower boundary of the search range. Must be less than `inTo`.

`inTo` [int](#)

The upper boundary of the search range. Must be greater than `inFrom`

`inMinDelta` [int](#)

The minimum allowable difference between successive input values, used to determine when the search should stop.

`invertingLogic` [bool](#)

Determines whether the logic is inverted. `false` means higher input values increase the likelihood of meeting the trip criteria. `true` means lower input values do.

`oneMeasurement` [Func](#)<Site<[int](#)>, Site<Tout>>

The action to execute for every measurement.

`outTripCriteria` [Func](#)<Tout, [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

`inNotFoundResult` [int](#)

The return value for the case when the trip criteria was never found.

Returns

`Site<int>`

The input value resulting in an output fulfilling the trip criteria and be closest to it's inflection point.
The worst case deviation from the exact input is (`inMinDelta`.

Type Parameters

Tout

The type of the device's output.

BinarySearch<Tout>(int, int, int, bool, Func<Site<int>, Site<Tout>>, Func<Tout, bool>, int, out Site<Tout>)

Performs an integer binary search between `inFrom` and `inTo` by executing `oneMeasurement` at each step. Determines the input resulting in an output closest to the trip criteria's inflection point. The number of steps executed does not exceed $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the input value closest to the transition point (within the specified `inMinDelta` resolution), but matching the criteria is reached. If no transition was found, the method returns `inNotFoundResult`.

```
Site<int> BinarySearch<Tout>(int inFrom, int inTo, int inMinDelta, bool invertingLogic,
Func<Site<int>, Site<Tout>> oneMeasurement, Func<Tout, bool> outTripCriteria, int
inNotFoundResult, out Site<Tout> outResult)
```

Parameters

inFrom [int](#)

The lower boundary of the search range. Must be less than `inTo`.

inTo [int](#)

The upper boundary of the search range. Must be greater than `inFrom`.

inMinDelta [int](#)

The minimum allowable difference between successive input values, used to determine when the search should stop.

invertingLogic [bool](#)

Determines whether the logic is inverted. `false` means higher input values increase the likelihood of meeting the trip criteria. `true` means lower input values do.

oneMeasurement [Func](#)<Site<[int](#)>, Site<Tout>>

The action to execute for every measurement.

outTripCriteria `Func<Tout, bool>`

A delegate indicating the output meets the condition required for the input value searched.

inNotFoundResult `int`

The return value for the case when the trip criteria was never found.

outResult `Site<Tout>`

Output - contains the output value of the input found.

Returns

`Site<int>`

The input value resulting in an output fulfilling the trip criteria and be closest to it's inflection point.

The worst case deviation from the exact input is (`inMinDelta`.

Type Parameters

Tout

The type of the device's output.

BinarySearch<Tout>(int, int, int, bool, Func<Site<int>, Site<Tout>>, Tout)

Performs an integer binary search between `inFrom` and `inTo` by executing `oneMeasurement` at each step. Determines the input resulting in an output closest to the target. The number of steps executed does not exceed $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the best input value (within the specified `inMinDelta` resolution) is reached.

```
Site<int> BinarySearch<Tout>(int inFrom, int inTo, int inMinDelta, bool invertingLogic,  
Func<Site<int>, Site<Tout>> oneMeasurement, Tout outTarget)
```

Parameters

inFrom `int`

The lower boundary of the search range. Must be less than `inTo`.

`inTo` [int](#)

The upper boundary of the search range. Must be greater than `inFrom`

`inMinDelta` [int](#)

The minimum allowable difference between successive input values, used to determine when the search should stop. Must be >0 .

`invertingLogic` [bool](#)

A flag indicating whether the output is inverted, meaning increasing output values are a result of decreasing input values.

`oneMeasurement` [Func](#)<`Site<int>`, `Site<Tout>`>

The action to execute for every measurement.

`outTarget` `Tout`

The (numeric) target output value for which the corresponding input condition is searched.

Returns

`Site<int>`

The input value resulting in an output closest to the target. The worst case delta to the exact input is $+/- (\text{inMinDelta} / 2)$ if it can be reached given the search range.

Type Parameters

`Tout`

The type of the device's output.

`BinarySearch<Tout>(int, int, int, bool, Func<Site<int>, Site<Tout>>, Tout, out Site<Tout>)`

Performs an integer binary search between `inFrom` and `inTo` by executing `oneMeasurement` at each step. Determines the input resulting in an output closest to the target. The number of steps executed does not

exceed $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the best input value (within the specified `inMinDelta` resolution) is reached.

```
Site<int> BinarySearch<Tout>(int inFrom, int inTo, int inMinDelta, bool invertingLogic,  
Func<Site<int>, Site<Tout>> oneMeasurement, Tout outTarget, out Site<Tout> outResult)
```

Parameters

`inFrom` [int](#)

The lower boundary of the search range. Must be less than `inTo`.

`inTo` [int](#)

The upper boundary of the search range. Must be greater than `inFrom`

`inMinDelta` [int](#)

The minimum allowable difference between successive input values, used to determine when the search should stop.

`invertingLogic` [bool](#)

A flag indicating whether the output is inverted, meaning increasing output values are a result of decreasing input values.

`oneMeasurement` [Func](#)<[Site<int](#)>, [Site<Tout](#)>

The action to execute for every measurement.

`outTarget` [Tout](#)

The (numeric) target output value for which the corresponding input condition is searched.

`outResult` [Site<Tout](#)>

Output - contains the output value for the input found.

Returns

[Site<int](#)>

The input value resulting in an output closest to the target. The worst case delta to the exact input is $+/- (\text{inMinDelta} / 2)$ if it can be reached given the search range.

Type Parameters

Tout

The type of the device's output.

LinearFullFromIncCount<Tin>(Tin, Tin, int, Action<Tin>)

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. The ramp ends after `inCount` measurements are completed.

```
void LinearFullFromIncCount<Tin>(Tin inFrom, Tin inIncrement, int inCount,  
Action<Tin> oneMeasurement)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inIncrement Tin

The input increment value for every step.

inCount int

The total number of steps to execute.

oneMeasurement Action<Tin>

The action to execute for every measurement.

Type Parameters

Tin

The type of the input condition for the device.

LinearFullFromToCount<Tin>(Tin, Tin, int, Action<Tin>)

Performs a linear search between `inFrom` and `inTo` with `inCount` inputs. Executes `oneMeasurement` at each step including the end points. The ramp ends after `inCount` measurements are completed.

```
Tin LinearFullFromToCount<Tin>(Tin inFrom, Tin inTo, int inCount,  
Action<Tin> oneMeasurement)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inTo` Tin

The end point of the linear input ramp.

`inCount` int

The number of equally spaced steps to execute, including both endpoints exactly.

`oneMeasurement` Action<Tin>

The action to execute for every measurement.

Returns

Tin

The calculated increment, for later use in the processing step (ignore if not needed).

Type Parameters

`Tin`

The type of the input condition for the device.

LinearFullFromToInc<Tin>(Tin, Tin, Tin, Action<Tin>)

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. The ramp ends with the last input less or equal `inTo`.

```
void LinearFullFromToInc<Tin>(Tin inFrom, Tin inTo, Tin inIncrement,  
Action<Tin> oneMeasurement)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inTo Tin

The end point of the linear input ramp.

inIncrement Tin

The input increment value for every step.

oneMeasurement Action<Tin>

The action to execute for every measurement.

Type Parameters

Tin

The type of the input condition for the device.

**LinearStopFromIncCount<Tin, Tout>(Tin, Tin, int, Tin, Tin,
Func<Tin, Site<Tout>>, Func<Tout, bool>)**

Performs a linear search from **inFrom** by increasing with **inIncrement**. Executes **oneMeasurement** at each step including the start point. Determines the first input meeting the **outTripCriteria**. The ramp stops prematurely when the trip criteria is met on all sites, or after **inCount** measurements are completed.

```
Site<Tin> LinearStopFromIncCount<Tin, Tout>(Tin inFrom, Tin inIncrement, int inCount,  
Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout,  
bool> outTripCriteria)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inIncrement `Tin`

The input increment value for every step.

inCount `int`

The total number of steps to execute.

inOffset `Tin`

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

oneMeasurement `Func`<`Tin`, `Site`<`Tout`>>

The action to execute for every measurement.

outTripCriteria `Func`<`Tout`, `bool`>

A delegate indicating the output meets the condition required for the input value searched.

Returns

`Site`<`Tin`>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

`Tin`

The type of the input condition for the device.

`Tout`

The type of the device's output.

`LinearStopFromIncCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>, out Site<int>)`

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or after `inCount` measurements are completed. Additionally provides the index of the input step found.

```
Site<Tin> LinearStopFromIncCount<Tin, Tout>(Tin inFrom, Tin inIncrement, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout, bool> outTripCriteria, out Site<int> tripIndex)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inIncrement` Tin

The input increment value for every step.

`inCount` [int](#)

The total number of steps to execute.

`inOffset` Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

`inNotFoundResult` Tin

The return value for the case when the trip criteria was never found.

`oneMeasurement` [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

`outTripCriteria` [Func](#)<Tout, [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

`tripIndex` Site<[int](#)>

Output - contains the index of the input step found.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromIncCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>, out Site<int>, out Site<Tout>)

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or after `inCount` measurements are completed. Additionally provides the index and output value of the input step found.

```
Site<Tin> LinearStopFromIncCount<Tin, Tout>(Tin inFrom, Tin inIncrement, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout, bool> outTripCriteria, out Site<int> tripIndex, out Site<Tout> tripOut)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inIncrement` Tin

The input increment value for every step.

inCount [int](#)

The total number of steps to execute.

inOffset [Tin](#)

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult [Tin](#)

The return value for the case when the trip criteria was never found.

oneMeasurement [Func](#)<[Tin](#), [Site<Tout>](#)>

The action to execute for every measurement.

outTripCriteria [Func](#)<[Tout](#), [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

tripIndex [Site<int](#)>

Output - contains the index of the input step found.

tripOut [Site<Tout>](#)

Output - contains the output value of the input step found.

Returns

[Site<Tin>](#)

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

`LinearStopFromIncCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Tout)`

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or after `inCount` measurements are completed.

```
Site<Tin> LinearStopFromIncCount<Tin, Tout>(Tin inFrom, Tin inIncrement, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inIncrement` Tin

The input increment value for every step.

`inCount` [int](#)

The total number of steps to execute.

`inOffset` Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

`inNotFoundResult` Tin

The return value for the case when the trip criteria was never found.

`oneMeasurement` [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

`outTarget` Tout

The (numeric) target output value to be searched.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromIncCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Tout, out Site<int>)

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or after `inCount` measurements are completed. Additionally provides the index of the input step found.

```
Site<Tin> LinearStopFromIncCount<Tin, Tout>(Tin inFrom, Tin inIncrement, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget, out Site<int> closestIndex)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inIncrement Tin

The input increment value for every step.

inCount int

The total number of steps to execute.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

oneMeasurement `Func<Tin, Site<Tout>>`

The action to execute for every measurement.

outTarget `Tout`

The (numeric) target output value to be searched.

closestIndex `Site<int>`

Output - contains the index of the input step found.

Returns

`Site<Tin>`

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

`LinearStopFromIncCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Tout, out Site<int>, out Site<Tout>)`

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or after `inCount` measurements are completed. Additionally provides the index and output value of the input step found.

```
Site<Tin> LinearStopFromIncCount<Tin, Tout>(Tin inFrom, Tin inIncrement, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget, out Site<int> closestIndex, out Site<Tout> closestOut)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inIncrement Tin

The input increment value for every step.

inCount int

The total number of steps to execute.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

oneMeasurement Func<Tin, Site<Tout>>

The action to execute for every measurement.

outTarget Tout

The (numeric) target output value to be searched.

closestIndex Site<int>

Output - contains the index of the input step found.

closestOut Site<Tout>

Output - contains the output value of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>)

Performs a linear search between `inFrom` and `inTo` with `inCount` inputs. Executes `oneMeasurement` at each step including the end points. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or after `inCount` measurements are completed.

```
Site<Tin> LinearStopFromToCount<Tin, Tout>(Tin inFrom, Tin inTo, int inCount, Tin  
inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout,  
bool> outTripCriteria)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inTo Tin

The end point of the linear input ramp.

inCount int

The number of equally spaced steps to execute, including both endpoints exactly.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

oneMeasurement `Func<Tin, Site<Tout>>`

The action to execute for every measurement.

outTripCriteria `Func<Tout, bool>`

A delegate indicating the output meets the condition required for the input value searched.

Returns

`Site<Tin>`

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

`Tin`

The type of the input condition for the device.

`Tout`

The type of the device's output.

`LinearStopFromToCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>, out Site<int>)`

Performs a linear search between `inFrom` and `inTo` with `inCount` inputs. Executes `oneMeasurement` at each step including the end points. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or after `inCount` measurements are completed. Additionally provides the index of the input step found.

```
Site<Tin> LinearStopFromToCount<Tin, Tout>(Tin inFrom, Tin inTo, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout, bool> outTripCriteria, out Site<int> tripIndex)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inTo Tin

The end point of the linear input ramp.

inCount [int](#)

The number of equally spaced steps to execute, including both endpoints exactly.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

oneMeasurement [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

outTripCriteria [Func](#)<Tout, [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

tripIndex Site<[int](#)>

Output - contains the index of the input step found.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>, out Site<int>, out Site<Tout>)

Performs a linear search between `inFrom` and `inTo` with `inCount` inputs. Executes `oneMeasurement` at each step including the end points. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or after `inCount` measurements are completed. Additionally provides the index and output value of the input step found.

```
Site<Tin> LinearStopFromToCount<Tin, Tout>(Tin inFrom, Tin inTo, int inCount, Tin inOffset,
Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout, bool>
outTripCriteria, out Site<int> tripIndex, out Site<Tout> tripOut)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inTo` Tin

The end point of the linear input ramp.

`inCount` [int](#)

The number of equally spaced steps to execute, including both endpoints exactly.

`inOffset` Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

`inNotFoundResult` Tin

The return value for the case when the trip criteria was never found.

`oneMeasurement` [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

outTripCriteria `Func<Tout, bool>`

A delegate indicating the output meets the condition required for the input value searched.

tripIndex `Site<int>`

Output - contains the index of the input step found.

tripOut `Site<Tout>`

Output - contains the output value of the input step found.

Returns

`Site<Tin>`

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Tout)

Performs a linear search between `inFrom` and `inTo` with `inCount` inputs. Executes `oneMeasurement` at each step including the end points. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or after `inCount` measurements are completed.

```
Site<Tin> LinearStopFromToCount<Tin, Tout>(Tin inFrom, Tin inTo, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget)
```

Parameters

inFrom `Tin`

The starting point of the linear input ramp.

inTo `Tin`

The end point of the linear input ramp.

inCount `int`

The number of equally spaced steps to execute, including both endpoints exactly.

inOffset `Tin`

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

oneMeasurement `Func`<`Tin`, `Site`<`Tout`>>

The action to execute for every measurement.

outTarget `Tout`

The (numeric) target output value to be searched.

Returns

`Site`<`Tin`>

The input value resulting in an output closest to the target.

Type Parameters

`Tin`

The type of the input condition for the device.

`Tout`

The type of the device's output.

`LinearStopFromToCount<Tin, Tout>(Tin inFrom, Tin inTo, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget, out Site<int> closestIndex)`

Performs a linear search between `inFrom` and `inTo` with `inCount` inputs. Executes `oneMeasurement` at each step including the end points. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or after `inCount` measurements are completed. Additionally provides the index of the input step found.

```
Site<Tin> LinearStopFromToCount<Tin, Tout>(Tin inFrom, Tin inTo, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget, out Site<int> closestIndex)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inTo` Tin

The end point of the linear input ramp.

`inCount` [int](#)

The number of equally spaced steps to execute, including both endpoints exactly.

`inOffset` Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

`inNotFoundResult` Tin

The return value for the case when the trip criteria was never found.

`oneMeasurement` [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

`outTarget` Tout

The (numeric) target output value to be searched.

`closestIndex` Site<[int](#)>

Output - contains the index of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Tout, out Site<int>, out Site<Tout>)

Performs a linear search between `inFrom` and `inTo` with `inCount` inputs. Executes `oneMeasurement` at each step including the end points. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or after `inCount` measurements are completed. Additionally provides the index and output value of the input step found.

```
Site<Tin> LinearStopFromToCount<Tin, Tout>(Tin inFrom, Tin inTo, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget, out Site<int> closestIndex, out Site<Tout> closestOut)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inTo` Tin

The end point of the linear input ramp.

`inCount` int

The number of equally spaced steps to execute, including both endpoints exactly.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

oneMeasurement [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

outTarget Tout

The (numeric) target output value to be searched.

closestIndex Site<[int](#)>

Output - contains the index of the input step found.

closestOut Site<Tout>

Output - contains the output value of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

`LinearStopFromToInc<Tin, Tout>(Tin, Tin, Tin, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>)`

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or with the last input less or equal `inTo`.

```
Site<Tin> LinearStopFromToInc<Tin, Tout>(Tin inFrom, Tin inTo, Tin inIncrement, Tin  
inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout,  
bool> outTripCriteria)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inTo` Tin

The end point of the linear input ramp.

`inIncrement` Tin

The input increment value for every step.

`inOffset` Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

`inNotFoundResult` Tin

The return value for the case when the trip criteria was never found.

`oneMeasurement` [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

`outTripCriteria` [Func](#)<Tout, [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToInc<Tin, Tout>(Tin, Tin, Tin, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>, out Site<int>)

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or with the last input less or equal `inTo`. Additionally provides the index of the input step found.

```
Site<Tin> LinearStopFromToInc<Tin, Tout>(Tin inFrom, Tin inTo, Tin inIncrement, Tin  
inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout, bool>  
outTripCriteria, out Site<int> tripIndex)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inTo Tin

The end point of the linear input ramp.

inIncrement Tin

The input increment value for every step.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

oneMeasurement `Func<Tin, Site<Tout>>`

The action to execute for every measurement.

outTripCriteria `Func<Tout, bool>`

A delegate indicating the output meets the condition required for the input value searched.

tripIndex `Site<int>`

Output - contains the index of the input step found.

Returns

`Site<Tin>`

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

`LinearStopFromToInc<Tin, Tout>(Tin, Tin, Tin, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>, out Site<int>, out Site<Tout>)`

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or with the last input less or equal `inTo`. Additionally provides the index and output value of the input step found.

```
Site<Tin> LinearStopFromToInc<Tin, Tout>(Tin inFrom, Tin inTo, Tin inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout, bool> outTripCriteria, out Site<int> tripIndex, out Site<Tout> tripOut)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inTo Tin

The end point of the linear input ramp.

inIncrement Tin

The input increment value for every step.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

oneMeasurement [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

outTripCriteria [Func](#)<Tout, bool>

A delegate indicating the output meets the condition required for the input value searched.

tripIndex Site<int>

Output - contains the index of the input step found.

tripOut Site<Tout>

Output - contains the output value of the input step found.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToInc<Tin, Tout>(Tin, Tin, Tin, Tin, Tin, Func<Tin, Site<Tout>>, Tout)

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or with the last input less or equal `inTo`.

```
Site<Tin> LinearStopFromToInc<Tin, Tout>(Tin inFrom, Tin inTo, Tin inIncrement, Tin  
inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inTo Tin

The end point of the linear input ramp.

inIncrement Tin

The input increment value for every step.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

oneMeasurement `Func<Tin, Site<Tout>>`

The action to execute for every measurement.

outTarget `Tout`

The (numeric) target output value to be searched.

Returns

`Site<Tin>`

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

`LinearStopFromToInc<Tin, Tout>(Tin, Tin, Tin, Tin, Tin, Func<Tin, Site<Tout>>, Tout, out Site<int>)`

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or with the last input less or equal `inTo`. Additionally provides the index of the input step found.

```
Site<Tin> LinearStopFromToInc<Tin, Tout>(Tin inFrom, Tin inTo, Tin inIncrement, Tin  
inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget, out  
Site<int> closestIndex)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inTo Tin

The end point of the linear input ramp.

inIncrement Tin

The input increment value for every step.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

oneMeasurement [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

outTarget Tout

The (numeric) target output value to be searched.

closestIndex Site<[int](#)>

Output - contains the index of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToInc<Tin, Tout>(Tin, Tin, Tin, Tin, Tin, Func<Tin, Site<Tout>>, Tout, out Site<int>, out Site<Tout>)

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or with the last input less or equal `inTo`. Additionally provides the index and output value of the input step found.

```
Site<Tin> LinearStopFromToInc<Tin, Tout>(Tin inFrom, Tin inTo, Tin inIncrement, Tin  
inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget, out  
Site<int> closestIndex, out Site<Tout> closestOut)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inTo` Tin

The end point of the linear input ramp.

`inIncrement` Tin

The input increment value for every step.

`inOffset` Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

`inNotFoundResult` Tin

The return value for the case when the trip criteria was never found.

`oneMeasurement` [Func<Tin, Site<Tout>>](#)

The action to execute for every measurement.

outTarget `Tout`

The (numeric) target output value to be searched.

closestIndex `Site<int>`

Output - contains the index of the input step found.

closestOut `Site<Tout>`

Output - contains the output value of the input step found.

Returns

`Site<Tin>`

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

Interface ILib.IDatalog

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the Datalog branch.

```
public interface ILib.IDatalog
```

Methods

TestFunctional(Site<bool>, string)

Perform a functional datalog test.

```
void TestFunctional(Site<bool> result, string pattern = "")
```

Parameters

result Site<[bool](#)>

The result object to be datalogged.

pattern [string](#)

Optional. The pattern executed.

TestParametric(PinSite<double>, double, string)

Perform a parametric datalog test by using FlowLimits.

```
void TestParametric(PinSite<double> result, double forceValue = 0, string forceUnit = "")
```

Parameters

result PinSite<[double](#)>

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

Examples

This code performs a `PinSite<double>` measurement on a DC instrument and datalogs the result.

```
public void MeasureDcAndDatalog() {  
    Pins pins = new Pins("dcvi");  
    PinSite<double> result = TheLib.Acquire.Dc.Measure(pins);  
    TheLib.Datalog.TestParametric(result, 5 * V, "V");  
}
```

TestParametric(PinSite<int>, double, string)

Perform a parametric datalog test by using FlowLimits.

```
void TestParametric(PinSite<int> result, double forceValue = 0, string forceUnit = "")
```

Parameters

result `PinSite<int>`

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

Examples

This code performs a `PinSite<int>` measurement on a DC instrument and datalogs the result.

TestParametric(PinSite<Samples<double>>, double, string, bool)

Perform a parametric datalog test by using FlowLimits.

```
void TestParametric(PinSite<Samples<double>> result, double forceValue = 0, string forceUnit = "", bool sameLimitForAllSamples = false)
```

Parameters

`result` `PinSite<Samples<double>>`

The result object to be datalogged.

`forceValue` `double`

Optional. The force value applied for the result.

`forceUnit` `string`

Optional. The force value's unit.

`sameLimitForAllSamples` `bool`

Optional. Whether to use the same FlowLimit for all samples.

TestParametric(PinSite<Samples<int>>, double, string, bool)

Perform a parametric datalog test by using FlowLimits.

```
void TestParametric(PinSite<Samples<int>> result, double forceValue = 0, string forceUnit = "", bool sameLimitForAllSamples = false)
```

Parameters

`result` `PinSite<Samples<int>>`

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

sameLimitForAllSamples [bool](#)

Optional. Whether to use the same FlowLimit for all samples.

TestParametric(Site<double>, double, string)

Perform a parametric datalog test by using FlowLimits.

```
void TestParametric(Site<double> result, double forceValue = 0, string forceUnit = "")
```

Parameters

result Site<[double](#)>

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

TestParametric(Site<int>, double, string)

Perform a parametric datalog test by using FlowLimits.

```
void TestParametric(Site<int> result, double forceValue = 0, string forceUnit = "")
```

Parameters

result Site<[int](#)>

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

TestParametric(Site<Samples<double>>, double, string, bool)

Perform a parametric datalog test by using FlowLimits.

```
void TestParametric(Site<Samples<double>> result, double forceValue = 0, string forceUnit = "", bool sameLimitForAllSamples = false)
```

Parameters

result Site<Samplesdouble

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

sameLimitForAllSamples [bool](#)

Optional. Whether to use the same FlowLimit for all samples.

TestParametric(Site<Samples<int>>, double, string, bool)

Perform a parametric datalog test by using FlowLimits.

```
void TestParametric(Site<Samples<int>> result, double forceValue = 0, string forceUnit = "", bool sameLimitForAllSamples = false)
```

Parameters

result Site<Samples<[int](#)>>

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

sameLimitForAllSamples [bool](#)

Optional. Whether to use the same FlowLimit for all samples.

TestScanNetwork(ScanNetworkPatternResults, ScanNetworkDatalogOption)

Perform a flexible datalog test for ScanNetwork pattern results, with datalogging options set by [ScanNetworkDatalogOption](#).

```
void TestScanNetwork(ScanNetworkPatternResults result, ScanNetworkDatalogOption  
datalogOptions)
```

Parameters

result [ScanNetworkPatternResults](#)

The ScanNetwork pattern result object of type [ScanNetworkPatternResults](#)

datalogOptions [ScanNetworkDatalogOption](#)

Interface ILib.IExecute

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the Execute branch.

```
public interface ILib.IExecute
```

Properties

Digital

The accessor for the Digital branch.

```
ILib.IExecute.IDigital Digital { get; }
```

Property Value

[ILib.IExecute.IDigital](#)

ScanNetwork

The accessor for the ScanNetwork branch.

```
ILib.IExecute.IScanNetwork ScanNetwork { get; }
```

Property Value

[ILib.IExecute.IScanNetwork](#)

Search

The accessor for the Search branch.

```
ILib.IExecute.ISearch Search { get; }
```

Property Value

[ILib.IExecute.ISearch](#)

Methods

Wait(double, bool, double)

Waits for the given time by updating the SettleWait timer, or - optionally - enforces a static wait.

```
void Wait(double time, bool staticWait = false, double timeout = 0.1)
```

Parameters

time [double](#)

The wait time in seconds.

staticWait [bool](#)

Optional. Whether to enforce a static wait.

timeout [double](#)

Optional. The timeout.

Interface ILib.IExecute.IDigital

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the Digital branch.

```
public interface ILib.IExecute.IDigital
```

Methods

ContinueToConditionalStop(PatternInfo, Action)

Continues a pattern to the next conditional stop and executed the action.

```
void ContinueToConditionalStop(PatternInfo pattern, Action action)
```

Parameters

pattern [PatternInfo](#)

Pattern to be executed.

action [Action](#)

Action to be called at the stop.

ForcePatternHalt()

Stops the currently running non-threaded pattern.

```
void ForcePatternHalt()
```

ForcePatternHalt(PatternInfo)

Stops the given pattern. Works for both threaded and non-threaded patterns.

```
void ForcePatternHalt(PatternInfo patternInfo)
```

Parameters

patternInfo [PatternInfo](#)

Pattern to halt.

RunPattern(PatternInfo)

Starts the pattern burst for the given pattern and waits for it to complete. Equivalent to calling [Start Pattern\(PatternInfo\)](#) and [WaitPatternDone\(PatternInfo\)](#) in sequence.

```
void RunPattern(PatternInfo patternInfo)
```

Parameters

patternInfo [PatternInfo](#)

Pattern to run.

RunPattern(SiteVariant)

Starts the pattern burst for the given SiteVariant and waits for it to complete. Equivalent to calling [Start Pattern\(PatternInfo\)](#) and [WaitPatternDone\(PatternInfo\)](#) in sequence.

```
void RunPattern(SiteVariant sitePatterns)
```

Parameters

sitePatterns SiteVariant

Sites run pattern.

RunPatternConditionalStop(PatternInfo, int, Func<PatternInfo, int, List<PinSite<double>>>)

Runs a pattern executing the specified func action at each conditional stop.

```
List<PinSite<double>>> RunPatternConditionalStop(PatternInfo pattern, int numberofStops,  
Func<PatternInfo, int, List<PinSite<double>>> func)
```

Parameters

pattern [PatternInfo](#)

Pattern to be executed.

numberOfStops [int](#)

Number of stops in the pattern.

func [Func](#)<PatternInfo, int, List<PinSite<double>>>

Func action to be called at each stop.

Returns

[List](#)<PinSite<double>>

Concatenated list of all the measurements taken at each stop.

StartPattern(PatternInfo)

Starts the pattern burst for the given pattern without waiting for it to complete.

```
void StartPattern(PatternInfo patternInfo)
```

Parameters

patternInfo [PatternInfo](#)

Pattern to start.

StartPattern(SiteVariant)

Starts the pattern burst for the given SiteVariant without waiting for it to complete.

```
void StartPattern(SiteVariant sitePatterns)
```

Parameters

sitePatterns SiteVariant

Sites to start pattern.

WaitPatternDone(PatternInfo)

Waits for the given pattern to complete execution before returning. Works for both threaded and non-threaded patterns.

```
void WaitPatternDone(PatternInfo patternInfo)
```

Parameters

patternInfo [PatternInfo](#)

Pattern to wait for completion.

Interface ILib.IExecute.IScanNetwork

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the ScanNetwork branch.

```
public interface ILib.IExecute.IScanNetwork
```

Methods

RunDiagnosis(ScanNetworkPatternInfo, ScanNetworkPatternResults, bool)

Runs diagnosis reburst on failed core instances, which is obtained from the ScanNetwork pattern results.

```
void RunDiagnosis(ScanNetworkPatternInfo scanNetworkPattern, ScanNetworkPatternResults  
nonDiagnosisResults, bool concurrentDiagnosis = false)
```

Parameters

scanNetworkPattern [ScanNetworkPatternInfo](#)

The [ScanNetworkPatternInfo](#) Object that is associated with the ScanNetwork pattern(set).

nonDiagnosisResults [ScanNetworkPatternResults](#)

The acquired ScanNetwork pattern results which contains failed core list.

concurrentDiagnosis [bool](#)

Optional. Whether to perform diagnosis on multiple core instances concurrently per reburst.

RunPattern(ScanNetworkPatternInfo)

Runs the ScanNetwork pattern(set) and reburst if needed until all icl instances' pass/fail results are obtained.

```
void RunPattern(ScanNetworkPatternInfo scanNetworkPattern)
```

Parameters

scanNetworkPattern [ScanNetworkPatternInfo](#)

The [ScanNetworkPatternInfo](#) Object that is associated with the ScanNetwork pattern(set).

Interface ILib.IExecute.ISearch

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the Search branch.

```
public interface ILib.IExecute.ISearch
```

Methods

LinearFullProcess<Tin, Tout>(List<Site<Tout>>, Tin, Tin, Tin, Tin, Func<Tout, bool>)

Processes the measurements of a linear full search to find the device input condition satisfying the trip criteria on the output.

```
Site<Tin> LinearFullProcess<Tin, Tout>(List<Site<Tout>> outValues, Tin inFrom, Tin  
inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tout, bool> outTripCriteria)
```

Parameters

outValues [List](#)<Site<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

outTripCriteria Func<Tout, bool>

A delegate indicating the output meets the condition required for the input value searched.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(List<Site<Tout>>, Tin, Tin, Tin, Tin, Func<Tout, bool>, out Site<int>)

Processes the measurements of a linear full search to find the device input condition satisfying the trip criteria on the output. Additionally provides the index of the input step found.

```
Site<Tin> LinearFullProcess<Tin, Tout>(List<Site<Tout>> outValues, Tin inFrom, Tin  
inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tout, bool> outTripCriteria, out  
Site<int> tripIndex)
```

Parameters

outValues List<Site<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement `Tin`

The per-step increment of the linear input ramp.

inOffset `Tin`

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

outTripCriteria `Func<Tout, bool>`

A delegate indicating the output meets the condition required for the input value searched.

tripIndex `Site<int>`

Output - contains the index of the input step found.

Returns

`Site<Tin>`

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

`LinearFullProcess<Tin, Tout>(List<Site<Tout>>, Tin, Tin, Tin, Tin, Func<Tout, bool>, out Site<int>, out Site<Tout>)`

Processes the measurements of a linear full search to find the device input condition satisfying the trip criteria on the output. Additionally provides the index and output value of the input step found.

```
Site<Tin> LinearFullProcess<Tin, Tout>(List<Site<Tout>> outValues, Tin inFrom, Tin inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tout, bool> outTripCriteria, out Site<int> tripIndex, out Site<Tout> tripOut)
```

Parameters

outValues [List](#)<Site<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

outTripCriteria [Func](#)<Tout, bool>

A delegate indicating the output meets the condition required for the input value searched.

tripIndex Site<int>

Output - contains the index of the input step found.

tripOut Site<Tout>

Output - contains the output value of the input step found.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(List<Site<Tout>>, Tin, Tin, Tin, Tout)

Processes the measurements of a linear full search to find the device input condition resulting in an output closest to the (numeric) target.

```
Site<Tin> LinearFullProcess<Tin, Tout>(List<Site<Tout>> outValues, Tin inFrom, Tin  
inIncrement, Tin inOffset, Tout outTarget)
```

Parameters

outValues [List](#)<Site<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

outTarget Tout

The (numeric) target output value to be searched.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(List<Site<Tout>>, Tin, Tin, Tin, Tout, out Site<int>)

Processes the measurements of a linear full search to find the device input condition resulting in an output closest to the (numeric) target. Additionally provides the index of the input step found.

```
Site<Tin> LinearFullProcess<Tin, Tout>(List<Site<Tout>> outValues, Tin inFrom, Tin  
inIncrement, Tin inOffset, Tout outTarget, out Site<int> closestIndex)
```

Parameters

outValues [List](#)<Site<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

outTarget Tout

The (numeric) target output value to be searched.

closestIndex Site<[int](#)>

Output - contains the index of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(List<Site<Tout>>, Tin, Tin, Tin, Tout, out Site<int>, out Site<Tout>)

Processes the measurements of a linear full search to find the device input condition resulting in an output closest to the (numeric) target. Additionally provides the index and output value of the input step found.

```
Site<Tin> LinearFullProcess<Tin, Tout>(List<Site<Tout>> outValues, Tin inFrom,  
Tin inIncrement, Tin inOffset, Tout outTarget, out Site<int> closestIndex, out  
Site<Tout> closestOut)
```

Parameters

outValues [List](#)<Site<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

outTarget Tout

The (numeric) target output value to be searched.

closestIndex Site<[int](#)>

Output - contains the index of the input step found.

closestOut Site<Tout>

Output - contains the output value of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(Site<Samples<Tout>>, Tin, Tin, Tin, Func<Tout, bool>)

Processes the measurements of a linear full search to find the device input condition satisfying the trip criteria on the output.

```
Site<Tin> LinearFullProcess<Tin, Tout>(Site<Samples<Tout>> outValues, Tin inFrom, Tin inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tout, bool> outTripCriteria)
```

Parameters

outValues Site<Samples<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

outTripCriteria Func<Tout, bool>

A delegate indicating the output meets the condition required for the input value searched.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(Site<Samples<Tout>>, Tin, Tin, Tin, Func<Tout, bool>, out Site<int>)

Processes the measurements of a linear full search to find the device input condition satisfying the trip criteria on the output. Additionally provides the index of the input step found.

```
Site<Tin> LinearFullProcess<Tin, Tout>(Site<Samples<Tout>> outValues, Tin inFrom, Tin  
inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tout, bool> outTripCriteria, out  
Site<int> tripIndex)
```

Parameters

outValues Site<Samples<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

outTripCriteria Func<Tout, bool>

A delegate indicating the output meets the condition required for the input value searched.

tripIndex Site<int>

Output - contains the index of the input step found.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(Site<Samples<Tout>>, Tin, Tin, Tin, Func<Tout, bool>, out Site<int>, out Site<Tout>)

Processes the measurements of a linear full search to find the device input condition satisfying the trip criteria on the output. Additionally provides the index and output value of the input step found.

```
Site<Tin> LinearFullProcess<Tin, Tout>(Site<Samples<Tout>> outValues, Tin inFrom, Tin  
inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tout, bool> outTripCriteria, out  
Site<int> tripIndex, out Site<Tout> tripOut)
```

Parameters

outValues Site<Samples<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

outTripCriteria `Func<Tout, bool>`

A delegate indicating the output meets the condition required for the input value searched.

tripIndex `Site<int>`

Output - contains the index of the input step found.

tripOut `Site<Tout>`

Output - contains the output value of the input step found.

Returns

`Site<Tin>`

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(Site<Samples<Tout>>, Tin, Tin, Tin, Tout)

Processes the measurements of a linear full search to find the device input condition resulting in an output closest to the (numeric) target.

```
Site<Tin> LinearFullProcess<Tin, Tout>(Site<Samples<Tout>> outValues, Tin inFrom, Tin inIncrement, Tin inOffset, Tout outTarget)
```

Parameters

outValues Site<Samples<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

outTarget Tout

The (numeric) target output value to be searched.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(Site<Samples<Tout>>, Tin, Tin, Tin, Tout, out Site<int>)

Processes the measurements of a linear full search to find the device input condition resulting in an output closest to the (numeric) target. Additionally provides the index of the input step found.

```
Site<Tin> LinearFullProcess<Tin, Tout>(Site<Samples<Tout>> outValues, Tin inFrom, Tin  
inIncrement, Tin inOffset, Tout outTarget, out Site<int> closestIndex)
```

Parameters

outValues Site<Samples<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

outTarget Tout

The (numeric) target output value to be searched.

closestIndex Site<int>

Output - contains the index of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(Site<Samples<Tout>>, Tin, Tin, Tin, Tout, out Site<int>, out Site<Tout>)

Processes the measurements of a linear full search to find the device input condition resulting in an output closest to the (numeric) target. Additionally provides the index and output value of the input step found.

```
Site<Tin> LinearFullProcess<Tin, Tout>(Site<Samples<Tout>> outValues, Tin inFrom, Tin inIncrement, Tin inOffset, Tout outTarget, out Site<int> closestIndex, out Site<Tout> closestOut)
```

Parameters

outValues Site<Samples<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

outTarget Tout

The (numeric) target output value to be searched.

closestIndex Site<int>

Output - contains the index of the input step found.

closestOut Site<Tout>

Output - contains the output value of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

Interface ILib.ISetup

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the Setup branch.

```
public interface ILib.ISetup
```

Properties

Dc

The accessor for the Dc branch.

```
ILib.ISetup.IDc Dc { get; }
```

Property Value

[ILib.ISetup.IDc](#)

Digital

The accessor for the Digital branch.

```
ILib.ISetup.IDigital Digital { get; }
```

Property Value

[ILib.ISetup.IDigital](#)

LevelsAndTiming

The accessor for the LevelsAndTiming branch.

```
ILib.ISetup.ILevelsAndTiming LevelsAndTiming { get; }
```

Property Value

[ILib.ISetup.ILevelsAndTiming](#)

Interface ILib.ISetup.IDc

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the Dc branch.

```
public interface ILib.ISetup.IDc
```

Extension Methods

[CustomerExtensions.CustomerExtension\(ILib.ISetup.IDc, string, int\)](#).

Methods

Connect(Pins, bool)

Connects and optionally gates on/off the pins depending on its instrument feature (PPMU, DCVI, DCVS,...).

```
void Connect(Pins pins, bool gateOn = false)
```

Parameters

pins [Pins](#)

The pins to connect.

gateOn [bool](#)

Optional. Default no gate change, True for gate on the pins after connecting.

ConnectAllPins()

Connects all power and digital pins from level context.

```
void ConnectAllPins()
```

Disconnect(Pins, bool)

Disconnects and optionally gates on/off the pins depending on its instrument feature (PPMU, DCVI, DCVS,...). It will disconnect in HiZ mode rather than forcing 0V or 0A on VIs.

```
void Disconnect(Pins pins, bool gateOff = true)
```

Parameters

pins [Pins](#)

The pins to disconnect.

gateOff [bool](#)

Optional. Default gate off (HiZ) the pins before disconnecting, False no gate change.

Force(Pins, TLibOutputMode, double, double, double, bool)

Sets the force current, force voltage or high impedance of the pins.

```
void Force(Pins pins, TLibOutputMode mode, double forceValue, double forceRange, double clampValue, bool gateOn = true)
```

Parameters

pins [Pins](#)

The pins to force.

mode [TLibOutputMode](#)

The mode for forcing (e.g., Voltage or Current).

forceValue [double](#)

The value to force.

forceRange [double](#)

The range for force value.

`clampValue` [double](#)

When forcing Voltage it sets the current limit and when forcing Current it sets the voltage range.

`gateOn` [bool](#)

Optional. Default gate on the pins after after the settings, False no gate change.

Force(Pins[], TLibOutputMode[], double[], double[], double[], bool[])

Sets the force current, force voltage or high impedance of each element in pinGroups.

```
void Force(Pins[] pinGroups, TLibOutputMode[] modes, double[] forceValues, double[]  
forceRanges, double[] clampValues, bool[] gateOn = null)
```

Parameters

`pinGroups` [Pins](#)[]

Array of pin or pin groups.

`modes` [TLibOutputMode](#)[]

Array of the mode for each pin or pin group.

`forceValues` [double](#)[]

Array of force values for each pin or pin group.

`forceRanges` [double](#)[]

Array of force ranges for each pin or pin group.

`clampValues` [double](#)[]

Array of clamp values for each pin or pin group.

`gateOn` [bool](#)[]

Optional. Array of gate state for each pin or pin group, default gate on for all pin or pin group.

ForceHiZ(Pins)

Sets to High Impedance mode.

```
void ForceHiZ(Pins pins)
```

Parameters

pins [Pins](#)

The pins to set in HiZ.

ForceI(Pins, double, double, double?, bool, bool, double?)

Sets the Force Current and the range of a DC instrument. Assuming that the instrument was already setup for remaining parameters.

```
void ForceI(Pins pins, double forceCurrent, double clampHiV, double currentRange, double?  
voltageRange = null, bool outputModeCurrent = false, bool gateOn = true, double? clampLoV  
= null)
```

Parameters

pins [Pins](#)

The pins to force the current.

forceCurrent [double](#) ↗

The current to force.

clampHiV [double](#) ↗

Sets the voltage for voltage clamp high.

currentRange [double](#) ↗

Expected current to set the current range.

voltageRange [double](#) ↗?

Optional. Expected voltage to set the voltage range or to program the voltage for DCVS.

outputModeCurrent [bool](#) ↗

Optional. Sets to true to switch to force current mode (if the mode was not previously set).

gateOn [bool](#) ↗

Optional. Default gate on the pins after after the settings, False no gate change.

clampLoV [double](#)?

Optional. Sets the voltage for voltage clamp low.

ForceI(Pins, double, double?, double?, bool, bool)

Sets the Force current of a DC instrument. Assumes the instrument was already setup to the right modes.

```
void ForceI(Pins pins, double forceCurrent, double? clampHiV = null, double? clampLoV = null, bool outputModeCurrent = false, bool gateOn = true)
```

Parameters

pins [Pins](#)

The pins to force the current.

forceCurrent [double](#) ↗

The current to force.

clampHiV [double](#)?

Optional. Sets the voltage for voltage clamp high.

clampLoV [double](#)?

Optional. Sets the voltage for voltage clamp low.

outputModeCurrent [bool](#) ↗

Optional. Sets to true to switch to force current mode (if the mode was not previously set).

gateOn [bool](#) ↗

Optional. Default gate on the pins after after the settings, False no gate change.

ForceV(Pins, double, double, double?, bool, bool)

Programs the force Voltage, measure range and many other parameters of a DC instrument - Advanced method with additional parameters.

```
void ForceV(Pins pins, double forceVoltage, double clampCurrent, double voltageRange,  
double? currentRange = null, bool outputModeVoltage = false, bool gateOn = true)
```

Parameters

pins [Pins](#)

The pins to force the voltage.

forceVoltage [double](#) ↗

The force voltage value.

clampCurrent [double](#) ↗

Current clamp value.

voltageRange [double](#) ↗

Expected voltage to set the Voltage range, if required, else use the forceVoltage.

currentRange [double](#) ↗?

Optional. Expected current to set the current range.

outputModeVoltage [bool](#) ↗

Optional. Sets to true to switch to force voltage mode (if the mode was not previously set).

gateOn [bool](#) ↗

Optional. Default gate on the pins after after the settings, False no gate change.

ForceV(Pins, double, double?, bool, bool)

Programs the force Voltage of a DC instrument. Simplest Method: It assumes the instrument is already in the right mode (FI,FV) and required ranges.

```
void ForceV(Pins pins, double forceVoltage, double? clampCurrent = null, bool  
outputModeVoltage = false, bool gateOn = true)
```

Parameters

pins [Pins](#)

The pins to force the voltage.

forceVoltage [double](#)?

The force voltage that will be set.

clampCurrent [double](#)?

Optional. Current clamp value.

outputModeVoltage [bool](#)?

Optional. Sets to true to switch to force voltage mode (if the mode was not previously set).

gateOn [bool](#)?

Optional. Default gate on the pins after after the settings, False no gate change.

Modify(Pins, DcParameters)

Selectively program or modify any DC instrument parameter.

```
void Modify(Pins pins, DcParameters parameters)
```

Parameters

pins [Pins](#)

The pins to set.

parameters [DcParameters](#)

The object through which each parameter can be set.

**Modify(Pins, bool?, TLibOutputMode?, double?, double?,
double?, double?, double?, double?, Measure?, double?,
double?, double?, double?, double?, double?, double?, bool?,
bool?, double?, double?, double?, double?, double?, bool?,
double?, double?)**

Selectively program or modify any DC instrument parameter.

```
void Modify(Pins pins, bool? gate = null, TLibOutputMode? mode = null, double? voltage = null, double? voltageAlt = null, double? current = null, double? voltageRange = null, double? currentRange = null, double? forceBandwidth = null, Measure? meterMode = null, double? meterVoltageRange = null, double? meterCurrentRange = null, double? meterBandwidth = null, double? sourceFoldLimit = null, double? sinkFoldLimit = null, double? sourceOverloadLimit = null, double? sinkOverloadLimit = null, bool? voltageAltOutput = null, bool? bleederResistor = null, double? complianceBoth = null, double? compliancePositive = null, double? complianceNegative = null, double? clampHiV = null, double? clampLoV = null, bool? highAccuracy = null, double? settlingTime = null, double? hardwareAverage = null)
```

Parameters

pins [Pins](#)

The pins to set.

gate [bool](#)?

Optional. Sets the gate.

mode [TLibOutputMode](#)?

Optional. Sets the operating mode.

voltage [double](#)?

Optional. Sets the output voltage.

voltageAlt [double](#)?

Optional. Sets the alternate output voltage.

`current` [double](#)?

Optional. Sets the output current.

`voltageRange` [double](#)?

Optional. Sets the voltage range.

`currentRange` [double](#)?

Optional. Sets the current range.

`forceBandwidth` [double](#)?

Optional. Sets the output compensation bandwidth.

`meterMode` [Measure](#)?

Optional. Sets the meter mode.

`meterVoltageRange` [double](#)?

Optional. Sets the meter voltage range.

`meterCurrentRange` [double](#)?

Optional. Sets the meter current range.

`meterBandwidth` [double](#)?

Optional. Sets the meter filter.

`sourceFoldLimit` [double](#)?

Optional. Sets the source fold limit.

`sinkFoldLimit` [double](#)?

Optional. Sets the sink fold limit.

`sourceOverloadLimit` [double](#)?

Optional. Sets the source overload limit.

`sinkOverloadLimit` [double](#)?

Optional. Sets the sink overload limit.

voltageAltOutput [bool](#)?

Optional. Sets the output DAC used to force voltage (true for alternate or false for main).

bleederResistor [bool](#)?

Optional. Sets the bleeder resistor's connection state.

complianceBoth [double](#)?

Optional. Sets both compliance ranges.

compliancePositive [double](#)?

Optional. Sets the positive compliance range.

complianceNegative [double](#)?

Optional. Sets the negative compliance range.

clampHiV [double](#)?

Optional. Sets the high voltage clamp value.

clampLoV [double](#)?

Optional. Sets the low voltage clamp value.

highAccuracy [bool](#)?

Optional. Sets the enabled state of the high accuracy measure voltage.

settlingTime [double](#)?

Optional. Sets the required additional settling time for the high accuracy measure voltage mode.

hardwareAverage [double](#)?

Optional. Sets the meter hardware average value.

SetForceAndMeter(Pins, TLibOutputMode, double, double, double, Measure, double, bool)

Programs the force and the meter's measure parameters interface for the dc instruments.

```
void SetForceAndMeter(Pins pins, TLibOutputMode mode, double forceValue, double forceRange,  
double clampValue, Measure meterMode, double measureRange, bool gateOn = true)
```

Parameters

pins [Pins](#)

The pins to set force and meter parameters.

mode [TLibOutputMode](#)

Set the output mode to TlibOutputMode Voltage, Current or HiZ.

forceValue [double](#)

Force voltage or current value.

forceRange [double](#)

Voltage or current to set the force range.

clampValue [double](#)

Current or voltage clamp value. Note: For PPMU it programs either clampVHi or clampVLo depending if sourcing or sinking current.

meterMode [Measure](#)

Set the meter's measure mode to measure voltage or current.

measureRange [double](#)

Set the meter's measure range to the expected current or voltage.

gateOn [bool](#)

Optional. Default gate on the pins after after the settings, False no gate change.

SetMeter(Pins, Measure, double, double?, int?, double?)

Sets the measurement interface of the instruments DCVI and DCVS.

```
void SetMeter(Pins pins, Measure meterMode, double rangeValue, double? filterValue = null,  
int? hardwareAverage = null, double? outputRangeValue = null)
```

Parameters

pins [Pins](#)

The pins to set meter parameters.

meterMode [Measure](#)

Set the mode to measure Voltage or Current.

rangeValue [double](#)?

Current or Voltage range depending on the selected mode.

filterValue [double](#)?

Optional. Sets the filter value.

hardwareAverage [int](#)?

Optional. Sets the hardware average for the specified DCVI pins.

outputRangeValue [double](#)?

Optional. Current range for DCVS when you want to set the current mode - for other cases, ignore this.

SetMeter(Pins[], Measure[], double[], double[], int[], double[])

Sets the measurements interface of the instruments DCVI and DCVS.

```
void SetMeter(Pins[] pinGroups, Measure[] meterModes, double[] rangeValues, double[]  
filterValues = null, int[] hardwareAverages = null, double[] outputRangeValues = null)
```

Parameters

pinGroups [Pins\[\]](#)

Array of pin or pin groups.

meterModes [Measure](#)[]

Array of settings measurements mode voltage and current.

rangeValues [double](#)[][]

Array of current and voltage range depending on the selected modes.

filterValues [double](#)[][]

Optional. Array of filter values.

hardwareAverages [int](#)[][]

Optional. Array of hardware average for the specified DCVI pins.

outputRangeValues [double](#)[][]

Optional. Array of current range for DCSV when you want to set the current mode - for other cases, ignore this.

Interface ILib.ISetup.IDigital

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the Digital branch.

```
public interface ILib.ISetup.IDigital
```

Methods

Connect(Pins)

Connect digital pins to the digital driver and comparator

```
void Connect(Pins pins)
```

Parameters

pins [Pins](#)

Pins to be connected, must contain digital pins

Disconnect(Pins)

Disconnect digital pins from the digital driver and comparator

```
void Disconnect(Pins pins)
```

Parameters

pins [Pins](#)

Pins to be disconnected, must contain digital pins

FrequencyCounter(Pins, double, FreqCtrEventSrcSel, FreqCtrEventSlopeSel)

Configures the digital instrument frequency counter.

```
void FrequencyCounter(Pins pins, double measureWindow, FreqCtrEventSrcSel eventSource,  
                      FreqCtrEventSlopeSel eventSlope)
```

Parameters

pins [Pins](#)

Digital pin(s) to be measured.

measureWindow [double](#) ↗

Time to measure the frequency.

eventSource FreqCtrEventSrcSel

The frequency counters comparator threshold.

eventSlope FreqCtrEventSlopeSel

The frequency counters event slope.

ModifyPins(Pins, DigitalPinsParameters)

Selectively program or modify any digital instrument pins parameter.

```
void ModifyPins(Pins pins, DigitalPinsParameters parameters)
```

Parameters

pins [Pins](#)

The pins to set.

parameters [DigitalPinsParameters](#)

The object through which each parameter can be set.

ModifyPins(Pins, tlHSDMAAlarm?, tlAlarmBehavior?, bool?, bool?, ChInitState?, ChStartState?, bool?, bool?)

Selectively program or modify any digital instrument Pins parameter.

```
void ModifyPins(Pins pins, tlHSDMAAlarm? alarmType = null, tlAlarmBehavior? alarmBehavior = null, bool? disableCompare = null, bool? disableDrive = null, ChInitState? initState = null, ChStartState? startState = null, bool? calibrationExcluded = null, bool? calibrationHighAccuracy = null)
```

Parameters

pins [Pins](#)

The pins to set.

alarmType [tlHSDMAAlarm?](#)

Optional. Sets the alarm type for the specified pins.

alarmBehavior [tlAlarmBehavior?](#)

Optional. Sets the alarm behavior for the specified pins.

disableCompare [bool?](#)

Optional. Disables the comparators for the specified pins

disableDrive [bool?](#)

Optional. Disables the drivers for the specified pins

initState [ChInitState?](#)

Optional. Sets the initial state of the pins

startState [ChStartState?](#)

Optional. Sets the start state of the pins

calibrationExcluded [bool?](#)

Optional. Sets the specified pins to be excluded from job dependent calibration

calibrationHighAccuracy [bool?](#)

Optional. Enables or disables calibration high accuracy mode for the specified pins

ModifyPinsLevels(Pins, DigitalPinsLevelsParameters)

Selectively program or modify any digital instrument pins levels parameter.

```
void ModifyPinsLevels(Pins pins, DigitalPinsLevelsParameters parameters)
```

Parameters

pins [Pins](#)

The pins to set.

parameters [DigitalPinsLevelsParameters](#)

The object through which each parameter can be set.

ModifyPinsLevels(Pins, ChDiffPinLevel?, double?, TLibDiffLvlValType[], double[], tIDriverMode?, ChPinLevel?, double?, SiteDouble, PinListData)

Selectively program or modify any digital instrument pins levels parameter.

```
void ModifyPinsLevels(Pins pins, ChDiffPinLevel? differentialLevelsType = null, double? differentialLevelsValue = null, TLibDiffLvlValType[] differentialLevelsValuesType = null, double[] differentialLevelsValues = null, tIDriverMode? levelsDriverMode = null, ChPinLevel? levelsType = null, double? levelsValue = null, SiteDouble levelsValuePerSite = null, PinListData levelsValues = null)
```

Parameters

pins [Pins](#)

The pins to set.

differentialLevelsType ChDiffPinLevel?

Optional. Sets the differential levels type for the specified pins

differentialLevelsValue [double](#)?

Optional. Sets the specified differential pin level type for the specified pins

differentialLevelsValuesType [TLibDiffLvlValType](#)[]

Optional. Sets the differential levels values type for the specified pins

differentialLevelsValues [double](#)[][]

Optional. Sets the specified differential pin levels values type for the specified pins

levelsDriverMode tlDriverMode?

Optional. Sets the driver mode for the specified pins

levelsType ChPinLevel?

Optional. Sets the level type for the specified pins

levelsValue [double](#)?

Optional. Sets the value for the specified level type on the specified pins

levelsValuePerSite SiteDouble

Optional. Sets the value for the specified level type for the specified pins on each site

levelsValues PinListData

Optional. Sets the value for the specified level value for each specified site and each specified pin

ModifyPinsTiming(Pins, DigitalPinsTimingParameters)

Selectively program or modify any digital instrument pins timing parameter.

```
void ModifyPinsTiming(Pins pins, DigitalPinsTimingParameters parameters)
```

Parameters

pins [Pins](#)

The pins to set.

parameters [DigitalPinsTimingParameters](#)

The object through which each parameter can be set.

ModifyPinsTiming(Pins, double?, double?, bool?, string, chEdge?, bool?, double?, double?, string, double?, tlOffsetType?, double?, bool?, SiteLong, int?, SiteDouble, AutoStrobeEnableSel?, int?, int?, double?, double?, bool?, double?, FreqCtrEnableSel?, FreqCtrEventSlopeSel?, FreqCtrEventSrcSel?, double?)

Selectively program or modify any digital instrument pins timing parameter.

```
void ModifyPinsTiming(Pins pins, double? timingClockOffset = null, double? timingClockPeriod = null, bool? timingDisableAllEdges = null, string timingEdgeSet = null, chEdge? timingEdgeVal = null, bool? timingEdgeEnabled = null, double? timingEdgeTime = null, double? timingRefOffset = null, string timingSetup1xDiagnosticCapture = null, double? timingSrcSyncDataDelay = null, tlOffsetType? timingOffsetType = null, double? timingOffsetValue = null, bool? timingOffsetEnabled = null, SiteLong timingOffsetSelectedPerSite = null, int? timingOffsetValuePerSiteIndex = null, SiteDouble timingOffsetValuePerSiteValue = null, AutoStrobeEnableSel? autoStrobeEnabled = null, int? autoStrobeNumSteps = null, int? autoStrobeSamplesPerStep = null, double? autoStrobeStartTime = null, double? autoStrobeStepTime = null, bool? freeRunningClockEnabled = null, double? freeRunningClockFrequency = null, FreqCtrEnableSel? freqCtrEnable = null, FreqCtrEventSlopeSel? freqCtrEventSlope = null, FreqCtrEventSrcSel? freqCtrEventSource = null, double? freqCtrInterval = null)
```

Parameters

[pins Pins](#)

The pins to set.

[timingClockOffset double?](#)

Optional. Sets the offset value between a DQS bus and a DUT clock in a DDR Protocol Aware test program for the specified pins

[timingClockPeriod double?](#)

Optional. Sets the current value for the period for the specified clock pins

[timingDisableAllEdges bool?](#)

Optional. Disables all edges (drive and compare) for the specified pins

timingEdgeSet [string](#)?

Optional. Sets the edgeset name for the specified pins

timingEdgeVal chEdge?

Optional. Sets the timing edge for the specified pins

timingEdgeEnabled [bool](#)?

Optional. Sets the enabled state for the specified pins and timing edge

timingEdgeTime [double](#)?

Optional. Sets the edge value for the specified pins and timing edge

timingRefOffset [double](#)?

Optional. Sets the offset value between the specified source synchronous reference (clock) pin and its data pins

timingSetup1xDiagnosticCapture [string](#)?

Optional. Sets up special dual-bit diagnostic capture in CMEM fail capture (LFVM) memory using the 1X pin setup for the specified pins and Time Sets sheet name

timingSrcSyncDataDelay [double](#)?

Optional. Sets the strobe reference data delay for individual source synchronous data pins

timingOffsetType tIOffsetType?

Optional. Sets the timing offset type for the specified pins

timingOffsetValue [double](#)?

Optional. Sets the timing offset value for the specified pins

timingOffsetEnabled [bool](#)?

Optional. Sets the timing offset enabled state for the specified pins

timingOffsetSelectedPerSite SiteLong

Optional. Sets the active offset index value for the specified pins on each site

timingOffsetValuePerSiteIndex [int](#)?

Optional. Set the timing offset index value. The valid index range is 0-7

timingOffsetValuePerSiteValue SiteDouble

Optional. Sets the current value for the offset at a specific index location that is to be applied to the timing values for the specified pins on each site

autoStrobeEnabled AutoStrobeEnableSel?

Optional. Enable state of the AutoStrobe engine for the specified pins

autoStrobeNumSteps [int](#)?

Optional. Sets the number of steps on the AutoStrobe engines for the specified pins

autoStrobeSamplesPerStep [int](#)?

Optional. Sets the number of samples per step on the AutoStrobe engines for the specified pins

autoStrobeStartTime [double](#)?

Optional. Sets the start time on the AutoStrobe engines for the specified pins

autoStrobeStepTime [double](#)?

Optional. Sets the step time on the AutoStrobe engines for the specified pins

freeRunningClockEnabled [bool](#)?

Optional. Sets the enable state of the free-running clock for the specified pins

freeRunningClockFrequency [double](#)?

Optional. Sets the frequency of the free-running clock for the specified pins

freqCtrEnable FreqCtrEnableSel?

Optional. Sets the frequency counter's enable state for the specified pins

freqCtrEventSlope FreqCtrEventSlopeSel?

Optional. Sets the frequency counter's event slope for the specified pins

freqCtrEventSource FreqCtrEventSrcSel?

Optional. Sets the frequency counter's event source for the specified pins

`freqCtrInterval` `double`?

Optional. Sets the duration of time to capture the frequency counter data for the specified pins

ReadAll()

Configures the tester to read all the vector data using HRAM.

```
void ReadAll()
```

ReadFails()

Configures the tester to read the failing vector data using HRAM.

```
void ReadFails()
```

ReadHram(int, CaptType, TrigType, bool, int)

Configures the tester for HRAM read back.

```
void ReadHram(int captureLimit, CaptType captureType, TrigType triggerType, bool  
waitForEvent, int preTriggerCycleCount)
```

Parameters

`captureLimit` `int`

Maximum number of vectors to be captured

`captureType` `CaptType`

Cycle type to be captured by HRAM, options are all, fail or stv

`triggerType` `TrigType`

Type of trigger for capture cycles, options are fail, first or never

`waitForEvent` `bool`

Sets whether the trigger waits for a cycle, vector or loop event

`preTriggerCycleCount int`

The number of cycles to capture before the trigger cycle.

ReadStoredVectors()

Configures the tester to read the data from vectors containing the STV statement.

`void ReadStoredVectors()`

Interface ILib.ISetup.ILevelsAndTiming

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

The interface for the LevelsAndTiming branch.

```
public interface ILib.ISetup.ILevelsAndTiming
```

Methods

Apply(bool, bool, bool)

Apply Connections, Levels and Timing, in either powered or unpowered mode.

```
void Apply(bool connectAllPins = false, bool unpowered = false, bool levelRampSequence = false)
```

Parameters

`connectAllPins` [bool](#)

Optional. If true: Connect all pins.

`unpowered` [bool](#)

Optional. If true: power down instruments and power supplies before connecting all pins.

`levelRampSequence` [bool](#)

Optional. If true: will ramp all levels with a predefined slew rate and sequence.

ApplyWithPinStates(bool, bool, bool, Pins, Pins, Pins)

Apply Connections, Level, Timing and set init states, in either powered or unpowered mode.

```
void ApplyWithPinStates(bool connectAllPins = false, bool unpowered = false, bool levelRampSequence = false, Pins initPinsHi = null, Pins initPinsLo = null, Pins initPinsHiZ
```

= `null`)

Parameters

`connectAllPins` [bool](#)

Optional. If true: Connect all pins.

`unpowered` [bool](#)

Optional. If true: power down instruments and power supplies before connecting all pins.

`levelRampSequence` [bool](#)

Optional. If true: will ramp all levels with a predefined slew rate and sequence.

`initPinsHi` [Pins](#)

Optional. Pin or pingroup to initialize to drive state high.

`initPinsLo` [Pins](#)

Optional. Pin or pingroup to initialize to drive state low.

`initPinsHiZ` [Pins](#)

Optional. Pin or pingroup to initialize to drive state tri-state.

Interface ILib.IValidate

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

```
public interface ILib.IValidate
```

Methods

Dc(Pins, bool?, TLibOutputMode?, double?, double?, double?,
double?, double?, double?, Measure?, double?, double?,
double?, double?, double?, double?, bool?, bool?,
double?, double?, double?, double?, bool?, double?,
double?)

Validate arguments within Test Method Templates.

```
void Dc(Pins pins, bool? gate = null, TLibOutputMode? mode = null, double? voltage = null,  
double? voltageAlt = null, double? current = null, double? voltageRange = null, double?  
currentRange = null, double? forceBandwidth = null, Measure? meterMode = null, double?  
meterVoltageRange = null, double? meterCurrentRange = null, double? meterBandwidth = null,  
double? sourceFoldLimit = null, double? sinkFoldLimit = null, double? sourceOverloadLimit =  
null, double? sinkOverloadLimit = null, bool? voltageAltOutput = null, bool? bleederResistor  
= null, double? complianceBoth = null, double? compliancePositive = null, double?  
complianceNegative = null, double? clampHiV = null, double? clampLoV = null, bool?  
highAccuracy = null, double? settlingTime = null, double? hardwareAverage = null)
```

Parameters

pins [Pins](#)

Pins parameter to be validated.

gate [bool](#)?

Optional. Gate parameter to be validated.

mode [TLibOutputMode](#)

Optional. Mode parameter to be validated.

voltage [double](#)?

Optional. Voltage parameter to be validated.

voltageAlt [double](#)?

Optional. VoltageAlt parameter to be validated.

current [double](#)?

Optional. Current parameter to be validated.

voltageRange [double](#)?

Optional. Voltage Range parameter to be validated.

currentRange [double](#)?

Optional. Current Range parameter to be validated.

forceBandwidth [double](#)?

Optional. Force Bandwidth parameter to be validated.

meterMode [Measure](#)?

Optional. Meter Mode parameter to be validated.

meterVoltageRange [double](#)?

Optional. Meter Voltage Range parameter to be validated.

meterCurrentRange [double](#)?

Optional. Meter Current Range parameter to be validated.

meterBandwidth [double](#)?

Optional. Meter Bandwidth parameter to be validated.

sourceFoldLimit [double](#)?

Optional. Source Fold Limit parameter to be validated.

sinkFoldLimit [double](#)?

Optional. Sink Fold Limit parameter to be validated.

sourceOverloadLimit [double](#)?

Optional. Source Overload Limit parameter to be validated.

sinkOverloadLimit [double](#)?

Optional. Sink Overload Limit parameter to be validated.

voltageAltOutput [bool](#)?

Optional. Voltage Alt Output parameter to be validated.

bleederResistor [bool](#)?

Optional. Bleeder Resistor parameter to be validated.

complianceBoth [double](#)?

Optional. Compliance Both parameter to be validated.

compliancePositive [double](#)?

Optional. Compliance Positive parameter to be validated.

complianceNegative [double](#)?

Optional. Compliance Negative parameter to be validated.

clampHiV [double](#)?

Optional. Clamp High V parameter to be validated.

clampLoV [double](#)?

Optional. Clamp Low V parameter to be validated.

highAccuracy [bool](#)?

Optional. High Accuracy parameter to be validated.

settlingTime [double](#)?

Optional. Settling Time parameter to be validated.

hardwareAverage [double](#)?

Optional. Hardware Average parameter to be validated.

Enum<T>(string, string, out T)

Checks if the provided string can be parsed to the specified enum type and create the enum value.

```
bool Enum<T>(string value, string argumentName, out T enumValue) where T : struct, Enum
```

Parameters

value [string](#)

The string to parse for. Case-Insensitive, specify only the enum member part.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

enumValue [T](#)

The enumeration value to output in the event that the provided value was successfully parsed in the provided enumeration.

Returns

[bool](#)

[true](#) if the **value** was found within the provided Enumeration and successfully parsed; otherwise, [false](#).

Type Parameters

T

The existing Enumeration to be parsed.

Fail(string, string)

Raises an unconditional validation error.

```
void Fail(string problemReasonResolutionMessage, string argumentName)
```

Parameters

problemReasonResolutionMessage [string](#)

Validation failure message clearly describing the problem, reason, and resolution message.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

GreaterOrEqual<T>(T, T, string)

Checks if a numeric value is greater or equal to a bound.

```
bool GreaterOrEqual<T>(T value, T boundary, string argumentName) where T : IComparable<T>
```

Parameters

value T

The value to be checked against the provided boundary.

boundary T

The boundary that the value must be greater than or equal to.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

Returns

[bool](#)

[true](#) if the **value** is greater than or equal to the provided boundary; otherwise, [false](#).

Type Parameters

T

The type specified for the provided parameters.

GreaterThan<T>(T, T, string)

Checks if a numeric value is greater than the provided boundary.

```
bool GreaterThan<T>(T value, T boundary, string argumentName) where T : IComparable<T>
```

Parameters

value T

The value to be checked against the provided boundary.

boundary T

The boundary that the value must be greater than.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

Returns

[bool](#)

[true](#) if the 'value' is greater than the provided boundary; otherwise, [false](#).

Type Parameters

T

The type specified for the provided parameters.

InRange<T>(T, T, T, string)

Checks if a numeric value is within a range (including).

```
bool InRange<T>(T value, T from, T to, string argumentName) where T : IComparable<T>
```

Parameters

value `T`

The value to be checked against the provided upper and lower threshold.

from `T`

The lower threshold.

to `T`

The upper threshold.

argumentName `string`

The argument name used to indicate to IG-XL which test instance parameter failed.

Returns

`bool`

`true` if the **value** is within the provided range; otherwise, `false`.

Type Parameters

`T`

The type specified for the provided parameters.

`IsTrue(bool, string, string)`

Checks for whether `condition == true`.

```
bool IsTrue(bool condition, string problemReasonResolutionMessage, string argumentName)
```

Parameters

condition `bool`

Condition to be checked.

problemReasonResolutionMessage `string`

Validation failure message describing the `problem`, `reason`, and `resolution`.

`argumentName` [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

Returns

[bool](#)

[true](#) if the condition == [true](#); Otherwise, [false](#).

LessOrEqual<T>(T, T, string)

Checks if a numeric value is less or equal to a bound.

```
bool LessOrEqual<T>(T value, T boundary, string argumentName) where T : IComparable<T>
```

Parameters

`value` T

The value to be checked against the provided boundary.

`boundary` T

The boundary that the value must be greater than or equal to.

`argumentName` [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

Returns

[bool](#)

[true](#) if the `value` is less than or equal to the provided boundary; otherwise, [false](#).

Type Parameters

T

The type specified for the provided parameters.

LessThan<T>(T, T, string)

Checks if a numeric value is less than the provided boundary.

```
bool LessThan<T>(T value, T boundary, string argumentName) where T : IComparable<T>
```

Parameters

value T

The value to be checked against the provided boundary.

boundary T

The boundary that the value must be less than.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

Returns

[bool](#)

[true](#) if the **value** is less than the provided boundary; otherwise, [false](#).

Type Parameters

T

The type specified for the provided parameters.

MethodHandle<T>(string, string, out MethodHandle<T>)

Checks for valid method handle spec and creates the object.

```
bool MethodHandle<T>(string fullyQualifiedName, string argumentName, out MethodHandle<T>
method) where T : Delegate
```

Parameters

fullyQualifiedNames [string](#)

Fully qualified name of the method to be checked.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

method [MethodHandle](#)<T>

Delegate to be created if the fullyQualifiedNames is found to be a valid method handle spec.

Returns

[bool](#)

[true](#) if the **fullyqualifiedNames** was found to match an existing method and the new delegate was created; otherwise, [false](#).

Type Parameters

T

The target delegate and it's accompanying parameter types.

MultiCondition<T>(string, Func<string, T>, string, out T[], int?)

Checks multi-condition validity and creates the data array.

```
bool MultiCondition<T>(string csv, Func<string, T> parser, string argumentName, out T[] conditions, int? referenceCount = null)
```

Parameters

csv [string](#)

Comma separated values to split and parse.

parser [Func](#)<[string](#), T>

Delegate used to parse comma separated list.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

conditions [T\[\]](#)

Output Array of parsed values sourced from the provided string.

referenceCount [int](#)?

Optional. If specified, the reference count to verify. Will report an error if the resulting array size is >1 (SingleCondition) and does not match (MultiCondition).

Returns

[bool](#)

[true](#) if the output array was successfully created; otherwise, [false](#).

Type Parameters

T

The target (output) type of the parser function.

MultiCondition<TEnum>(string, string, out TEnum[], int?)

Checks multi-condition validity and creates the data array.

```
bool MultiCondition<TEnum>(string csv, string argumentName, out TEnum[] conditions, int?  
referenceCount = null) where TEnum : struct, Enum
```

Parameters

csv [string](#)

Comma separated values to split and parse. Case-Insensitive, specify only the enum member part.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

conditions [TEnum\[\]](#)

Output Array of parsed values sourced from the provided string.

referenceCount [int](#)?

Optional. If specified, the reference count to verify. Will report an error if the resulting array size is >1 (SingleCondition) and does not match (MultiCondition).

Returns

[bool](#)

[true](#) if the output array was successfully created; otherwise, [false](#).

Type Parameters

[TEnum](#)

Pattern(Pattern, string, out PatternInfo, bool)

Checks for valid pattern spec and creates patternInfo object.

```
bool Pattern(Pattern pattern, string argumentName, out PatternInfo patternInfo, bool  
threading = true)
```

Parameters

pattern [Pattern](#)

The pattern file to check.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

patternInfo [PatternInfo](#)

A new [PatternInfo](#) object containing the provided pattern.

threading [bool](#)

Optional. Indicate whether threading should be used. Validation will fail if threading is not supported by the pattern.

Returns

[bool](#)

[true](#) if the **pattern** exists and creates the new PatternInfo() object; otherwise, [false](#).

Pins(PinList, string, out Pins)

Checks for C#RA supported pin spec and creates the object.

```
bool Pins(PinList pinlist, string argumentName, out Pins pins)
```

Parameters

pinlist PinList

The pinlist to be checked.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

pins [Pins](#)

A new [Pins](#) object containing all of the resolved pins.

Returns

[bool](#)

[true](#) if the **pinList** was comprised of valid pins and creates the **Pins()** object; otherwise, [false](#).

Interface IService

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

```
public interface IService
```

Methods

Reset()

Initialize the service. This is called by the API when the service is first used.

```
void Reset()
```

Interface ISetting

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

```
public interface ISetting
```

Methods

Apply()

Applies the setting to the hardware, in case and only for those pins that need it.

```
void Apply()
```

Diff()

Performs a diff on the setting. This is used to check if the setting is already applied to the hardware.

```
void Diff()
```

Dump()

Dumps the setting to the log output target.

```
void Dump()
```

Export(string)

Exports the setting to a file at the specified path. The file format is implementation specific, but should be human readable.

```
void Export(string path)
```

Parameters

path [string](#)

The file system path where the setups will be exported.

Init(InitMode)

Performs the specified initialization to the setting.

```
void Init(InitMode initMode)
```

Parameters

initMode [InitMode](#)

The init mode.

Interface ISetupService

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

```
public interface ISetupService : IService
```

Inherited Members

[IService.Reset\(\)](#)

Properties

AuditMode

Reads or sets audit mode (performs hardware reads to confirm cached data is correct).

```
bool AuditMode { get; set; }
```

Property Value

[bool](#) ↗

Count

Gets the number of setups contained in the SetupService.

```
int Count { get; }
```

Property Value

[int](#) ↗

RespectSettlingTimeDefault

Reads or sets the global default for respecting settling times when applying settings.

```
bool RespectSettlingTimeDefault { get; set; }
```

Property Value

[bool](#)

SettleWaitTimeOut

Reads or sets the global default for settling time-out.

```
double SettleWaitTimeOut { get; set; }
```

Property Value

[double](#)

VerboseMode

Reads or sets verbose mode (prints detailed information to log output target).

```
bool VerboseMode { get; set; }
```

Property Value

[bool](#)

Methods

Add(Setup)

Adds the specified setup to the SetupService.

```
void Add(Setup setup)
```

Parameters

`setup` [Setup](#)

The setup to add.

Apply(string)

Applies one or multiple named setups from the SetupService. If more than one is specified, the requested sequence is maintained. Does nothing if the name is null or empty.

```
void Apply(string setupNames)
```

Parameters

`setupNames` [string](#)

The setup(s) to be applied (CSV).

Diff(string)

Performs a diff of the specified setups against current hardware state and logs the differences to the output window.

```
void Diff(string setupNames)
```

Parameters

`setupNames` [string](#)

Dump()

Dumps all setups to the log output target.

```
void Dump()
```

Export(string, string)

Exports the specified setups to a file at the specified path. The file format is JSON. If the file already exists, it will be overwritten. If the path is null or empty, the setups will not be exported. An empty parameter `setupNames` will export all setups in the SetupService.

```
void Export(string path, string setupNames)
```

Parameters

`path` [string](#)

The file system path where the setups will be exported.

`setupNames` [string](#)

The setup(s) to be exported (CSV).

Import(string, bool)

Imports setups from a file at the specified path. If `overwrite` is true, setups with the same name are overwritten.

```
void Import(string path, bool overwrite = true)
```

Parameters

`path` [string](#)

The file system path from where the setups will be imported.

`overwrite` [bool](#)

Optional. If set to `true`, existing setups will be overwritten.

Init(InitMode)

Performs the specified initialization to all setups in the SetupService.

```
void Init(InitMode initMode)
```

Parameters

`initMode` [InitMode](#)

The init mode.

Log(string, int, int)

Logs a message with a specified severity level and color.

```
void Log(string message, int level, int rgb)
```

Parameters

`message` [string](#)

The message to be logged. Cannot be null or empty.

`level` [int](#)

The level of indentation for the message.

`rgb` [int](#)

The RGB color value used to display the message. Must be a valid RGB integer.

Remove(string)

Removes the setup with the specified setupName from the SetupService.

```
bool Remove(string setupName)
```

Parameters

`setupName` [string](#)

The setup to be removed.

Returns

bool

true if the **setup** is successfully found and removed; otherwise, false.

Interface IStorageService

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

```
public interface IStorageService : IService
```

Inherited Members

[IService.Reset\(\)](#)

Properties

Count

Gets the number of key/value pairs contained in the storage.

```
int Count { get; }
```

Property Value

[int](#)

Keys

Gets a collection containing the keys in the storage.

```
IEnumerable<string> Keys { get; }
```

Property Value

[IEnumerable<string>](#)

Methods

AddOrUpdate(string, object)

Adds a key/value pair to the storage if the key does not already exist, or updates the value for an existing key.

```
void AddOrUpdate(string key, object value)
```

Parameters

key [string](#)

The key to be added or whose value should be updated.

value [object](#)

The value of the element to add, or update.

ContainsKey(string)

Determines whether the storage contains the specified key.

```
bool ContainsKey(string key)
```

Parameters

key [string](#)

The key to locate in the storage.

Returns

[bool](#)

[true](#) if the storage contains an element with the specified key; otherwise, [false](#).

Remove(string)

Removes the value with the specified key from the storage.

```
bool Remove(string key)
```

Parameters

key [string](#)

The key of the element to remove.

Returns

[bool](#)

[true](#) if the element is successfully found and removed; otherwise [false](#).

TryGetValue<T>(string, out T)

Gets the type-safe value associated with the specified key.

```
bool TryGetValue<T>(string key, out T value)
```

Parameters

key [string](#)

The key of the value to get.

value [T](#)

When this methods returns, contains the value associated with the specified key, if the key is found; otherwise, the default value for the type of the value parameter. This parameter is passed uninitialized.

Returns

[bool](#)

[true](#) if the storage contains an element with the specified key; otherwise, [false](#).

Type Parameters

[T](#)

The type of object to retrieve.

Interface ITransactionConfig

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

```
public interface ITransactionConfig
```

Properties

Valid

```
bool Valid { get; }
```

Property Value

[bool](#) ↗

Interface ITransactionService

Namespace: [Csra.Interfaces](#)

Assembly: Csra.dll

```
public interface ITransactionService : IService
```

Inherited Members

[IService.Reset\(\)](#)

Methods

ConfigureHandler(ITransactionConfig)

Configures the handler for the current transaction type.

```
bool ConfigureHandler(ITransactionConfig config)
```

Parameters

config [ITransactionConfig](#)

Returns

[bool](#)

Execute(string, string)

Execute a transaction module, pushing data from the shadow register to the DUT (without reads).

```
void Execute(string module, string port = "")
```

Parameters

module [string](#)

The name of the transaction module.

port [string](#)

Optional. The name of the port to use for transaction, if empty default or first port is used.

ExecuteRead<T>(string, int, string)

Executes a transaction file, pushing data from the shadow register to the DUT and reading data from the DUT to the code.

```
List<Site<T>> ExecuteRead<T>(string module, int readCount, string port = "")
```

Parameters

module [string](#)

The name of the module.

readCount [int](#)

The number of results to read after execution.

port [string](#)

Optional. The name of the port to use for transaction, if empty default or first port is used.

Returns

[List](#)<Site<T>>

A list of values read from the DUT during execution.

Type Parameters

T

The type defined for this port.

ExpectRegisterPerSite<T>(string, Site<T>, string)

Checks if the register matches the expected data.

```
Site<bool> ExpectRegisterPerSite<T>(string register, Site<T> data, string port = "")
```

Parameters

register [string](#)

The name of the register.

data Site<T>

The data to be compared to the register.

port [string](#)

Optional. The name of the port to use for transaction, if empty default or first port is used.

Returns

Site<[bool](#)>

True if the register content matches the data.

Type Parameters

T

The type defined for this port.

ExpectRegister<T>(string, T, string)

Reads a register from the DUT and checks if the content matches the expected data.

```
Site<bool> ExpectRegister<T>(string register, T data, string port = "")
```

Parameters

register [string](#)

The name of the register.

data `T`

The data to be compared to the register.

port `string` ↗

Optional. The name of the port to use for transaction, if empty default or first port is used.

Returns

`Site<bool>`

True if the register content matches the data.

Type Parameters

`T`

The type defined for this port.

GetField<T>(string, string, string)

Retrieves a field from the shadow register.

```
Site<T> GetField<T>(string register, string field, string port = "")
```

Parameters

register `string` ↗

The name of the register.

field `string` ↗

The name of the field.

port `string` ↗

Optional. The name of the port to use for transaction, if empty default or first port is used.

Returns

Site<T>

The value of the field.

Type Parameters

T

The type defined for this port.

GetRegister<T>(string, string)

Retrieves a register from the shadow register.

```
Site<T> GetRegister<T>(string register, string port = "")
```

Parameters

register [string](#)

The name of the register.

port [string](#)

Optional. The name of the port to use for transaction, if empty default or first port is used.

Returns

Site<T>

The content of the register.

Type Parameters

T

The type defined for this port.

Init(TransactionType)

Initializes the transaction service for a specific transaction type. This must be called before any other transaction methods are used. Multiple transaction types can be initialized, but only one can be active at a time. To switch the active transaction type, call Init again with the desired type.

```
void Init(TransactionType transactionType)
```

Parameters

transactionType [TransactionType](#)

PullRegister(string, string)

Pulls a register from the DUT to the shadow register.

```
void PullRegister(string register, string port = "")
```

Parameters

register [string](#)

The name of the register.

port [string](#)

Optional. The name of the port to use for transaction, if empty default or first port is used.

PushRegister(string, string)

Pushes a register from the shadow register to the DUT.

```
void PushRegister(string register, string port = "")
```

Parameters

register [string](#)

The name of the register.

port [string](#)

Optional. The name of the port to use for transaction, if empty default or first port is used.

ReInitRegister(string, string)

Sets a single shadow register to it's default / init value.

```
void ReInitRegister(string register, string port = "")
```

Parameters

register [string](#)

The name of the register.

port [string](#)

Optional. The name of the port to use for transaction, if empty default or first port is used.

ReadRegister<T>(string, string)

Reads a register from the DUT.

```
Site<T> ReadRegister<T>(string register, string port = "")
```

Parameters

register [string](#)

The name of the register.

port [string](#)

Optional. The name of the port to use for transaction, if empty default or first port is used.

Returns

[Site<T>](#)

The content of the register read from the DUT.

Type Parameters

T

The type defined for this port.

RemoveHandler(TransactionType)

Removes the handler for the specified transaction type.

```
void RemoveHandler(TransactionType transactionType)
```

Parameters

transactionType [TransactionType](#)

SetFieldPerSite<T>(string, string, Site<T>, string)

Sets a field in the shadow register (site-specific).

```
void SetFieldPerSite<T>(string register, string field, Site<T> data, string port = "")
```

Parameters

register [string](#) ↗

The name of the register.

field [string](#) ↗

The name of the field.

data [Site<T>](#)

The data to be set.

port [string](#) ↗

Optional. The name of the port to use for transaction, if empty default or first port is used.

Type Parameters

T

The type defined for this port.

SetField<T>(string, string, T, string)

Sets a field in the shadow register (site-uniform).

```
void SetField<T>(string register, string field, T data, string port = "")
```

Parameters

register string ↗

The name of the register.

field string ↗

The name of the field.

data T

The data to be set.

port string ↗

Optional. The name of the port to use for transaction, if empty default or first port is used.

Type Parameters

T

The type defined for this port.

SetRegisterPerSite<T>(string, Site<T>, string)

Sets a register in the shadow register (site-specific).

```
void SetRegisterPerSite<T>(string register, Site<T> data, string port = "")
```

Parameters

register [string](#)

The name of the register.

data [Site<T>](#)

The data to be set.

port [string](#)

Optional. The name of the port to use for transaction, if empty default or first port is used.

Type Parameters

T

The type defined for this port.

SetRegister<T>(string, T, string)

Sets a register in the shadow register (site-uniform).

```
void SetRegister<T>(string register, T data, string port = "")
```

Parameters

register [string](#)

The name of the register.

data [T](#)

The data to be set.

port [string](#)

Optional. The name of the port to use for transaction, if empty default or first port is used.

Type Parameters

T

The type defined for this port.

WriteRegisterPerSite<T>(string, Site<T>, string)

Writes data to a register in the DUT (site-specific).

```
void WriteRegisterPerSite<T>(string register, Site<T> data, string port = "")
```

Parameters

register [string](#)

The name of the register.

data [Site<T>](#)

The data to be written to the register.

port [string](#)

Optional. The name of the port to use for transaction, if empty default or first port is used.

Type Parameters

T

The type defined for this port.

WriteRegister<T>(string, T, string)

Writes data to a register in the DUT (site-uniform).

```
void WriteRegister<T>(string register, T data, string port = "")
```

Parameters

register [string](#)

The name of the register.

data T

The data to be written to the register.

port [string](#)

Optional. The name of the port to use for transaction, if empty default or first port is used.

Type Parameters

T

The type defined for this port.

Namespace Csra.Services

Classes

Alert

AlertService - centralized info / warning / error alerting.

Behavior

BehaviorService - when logic gets a personality.

Setup

Services.Setup - centralized test configuration management.

Storage

StorageService - centralized persistent data storage.

Class Alert

Namespace: [Csra.Services](#)

Assembly: Csra.dll

AlertService - centralized info / warning / error alerting.

```
public class Alert : IAlertService, IService
```

Inheritance

[object](#) ← Alert

Implements

[IAlertService](#), [IService](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Constructors

Alert()

```
public Alert()
```

Properties

ErrorTarget

Reads or sets the output target for Error Alerts. Defaults to OutputWindow and Datalog, which cannot be disabled.

```
public AlertOutputTarget ErrorTarget { get; set; }
```

Property Value

[AlertOutputTarget](#)

InfoTarget

Reads or sets the output target for Info Alerts. Defaults to OutputWindow, which cannot be disabled.

```
public AlertOutputTarget InfoTarget { get; set; }
```

Property Value

[AlertOutputTarget](#)

LogTarget

Reads or sets the output target for Log Alerts. Defaults to OutputWindow, which cannot be disabled.

```
public AlertOutputTarget LogTarget { get; set; }
```

Property Value

[AlertOutputTarget](#)

OutputFile

Reads or sets the file path for the output file.

```
public string OutputFile { get; set; }
```

Property Value

[string](#)

TimeStamp

Reads or sets whether a time stamp is added to Info / Warning and Error Alerts.

```
public bool TimeStamp { get; set; }
```

Property Value

[bool](#)

WarningTarget

Reads or sets the output target for Warning Alerts. Defaults to OutputWindow and Datalog, which cannot be disabled.

```
public AlertOutputTarget WarningTarget { get; set; }
```

Property Value

[AlertOutputTarget](#)

Methods

Error(string, int, string)

Sends an Error Alert message to the selected output target(s). At test program runtime, this raises an exception to the IG-XL error handler. If called during validation, it fails validation and flags the error appropriately. Use this for non-recoverable conditions that require immediate and safe termination.

Note: The compiler does not recognize that this method never returns, so you may need extra checks to satisfy .NET Framework requirements.

```
public void Error(string error, int validationArgumentIndex = 0, string doNotSpecify = "")
```

Parameters

error [string](#)

The Error Alert message.

validationArgumentIndex [int](#)

Optional. The offending test instance argument index (one-based), if applicable when used in validation.

doNotSpecify [string](#)

DO NOT SPECIFY - the name of the calling method is automatically inserted by the compiler.

Error<TException>(string, int, string)

Sends an Error Alert message to the selected output target(s). At test program runtime, this raises a user-selectable exception to the IG-XL error handler. If called during validation, it fails validation and flags the error appropriately. Use this for non-recoverable conditions that require immediate and safe termination. Note: The compiler does not recognize that this method never returns, so you may need extra checks to satisfy .NET Framework requirements.

```
public void Error<TException>(string error, int validationArgumentIndex = 0, string  
doNotSpecify = "") where TException : Exception
```

Parameters

error [string](#)

The Error Alert message.

validationArgumentIndex [int](#)

Optional. The offending test instance argument index (one-based), if applicable when used in validation.

doNotSpecify [string](#)

DO NOT SPECIFY - the name of the calling method is automatically inserted by the compiler.

Type Parameters

TException

The type of the exception to be thrown.

Info(string, string)

Sends an Info Alert message to the selected output target(s). Use for positive / neutral information relevant to the user.

```
public void Info(string info, string doNotSpecify = "")
```

Parameters

info [string](#)

The Info Alert message.

doNotSpecify [string](#)

DO NOT SPECIFY - the name of the calling method is automatically inserted by the compiler.

Log(string, byte, byte, byte, bool)

Sends a Log Alert message to the selected output target(s).

```
public void Log(string message, byte red, byte green, byte blue, bool bold = false)
```

Parameters

message [string](#)

The message string to be logged.

red [byte](#)

The red component of a RGB color.

green [byte](#)

The green component of a RGB color.

blue [byte](#)

The blue component of a RGB color.

bold [bool](#)

Optional. Whether bold font is used (OutputWindow only, ignored elsewhere).

Log(string, ColorConstants, bool)

Sends a Log Alert message to the selected output target(s).

```
public void Log(string message, ColorConstants color = ColorConstants.Black, bool bold = false)
```

Parameters

message [string](#)

The message string to be logged.

color [ColorConstants](#)

Optional. The color to be used (OutputWindow only, ignored elsewhere).

bold [bool](#)

Optional. Whether bold font is used (OutputWindow only, ignored elsewhere).

Reset()

Initialize the service. This is called by the API when the service is first used.

```
public void Reset()
```

Warning(string, string)

Sends a Warning Alert message to the selected output target(s). Use for recoverable issues that may require attention.

```
public void Warning(string warning, string doNotSpecify = "")
```

Parameters

warning [string](#)

The Warning Alert message.

doNotSpecify [string](#)

DO NOT SPECIFY - the name of the calling method is automatically inserted by the compiler.

Class Behavior

Namespace: [Csra.Services](#)

Assembly: Csra.dll

BehaviorService - when logic gets a personality.

```
public class Behavior : IBehaviorService, IService
```

Inheritance

[object](#) ← Behavior

Implements

[IBehaviorService](#), [IService](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Properties

Features

Gets a collection containing all defined features.

```
public IEnumerable<string> Features { get; }
```

Property Value

[IEnumerable](#) <[string](#)>

FilePath

Gets/Sets the import / export file path.

```
public string FilePath { get; set; }
```

Property Value

[string](#) ↗

Methods

Export(string)

Writes all features to the specified file, or a previously defined [FilePath](#) if empty. Updates the [FilePath](#) setting.

```
public void Export(string filePath = "")
```

Parameters

[filePath](#) [string](#) ↗

Optional. The (relative or absolute) file path.

GetFeature<T>(string)

Reads a feature's value. Type must match the original definition, an exception is thrown otherwise.

```
public T GetFeature<T>(string feature)
```

Parameters

[feature](#) [string](#) ↗

The feature name.

Returns

T

The feature's value.

Type Parameters

The feature value's type.

Import(string)

Reads the specified file, or a previously defined [FilePath](#) if empty. Incrementally updates features with new values. Call [Services.Behavior.Reset\(\)](#) to clear all data first. Updates the [FilePath](#) setting.

```
public void Import(string filePath = "")
```

Parameters

filePath [string](#) ↗

Optional. The (relative or absolute) file path.

Reset()

Initialize the service. This is called by the API when the service is first used.

```
public void Reset()
```

SetFeature<T>(string, T)

Defines a feature's value. Creates a new entry, or updates an existing one if it already exists.

```
public void SetFeature<T>(string feature, T value)
```

Parameters

feature [string](#) ↗

The feature name.

value [T](#)

The feature's new value.

Type Parameters

T

The feature value's type.

Class Setup

Namespace: [Csra.Services](#)

Assembly: Csra.dll

Services.Setup - centralized test configuration management.

```
public class Setup : ISetupService, IService
```

Inheritance

[object](#) ← Setup

Implements

[ISetupService](#), [IService](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Properties

AuditMode

Reads or sets audit mode (performs hardware reads to confirm cached data is correct).

```
public bool AuditMode { get; set; }
```

Property Value

[bool](#)

Count

Gets the number of setups contained in the SetupService.

```
public int Count { get; }
```

Property Value

[int](#)

RespectSettlingTimeDefault

Reads or sets the global default for respecting settling times when applying settings.

```
public bool RespectSettlingTimeDefault { get; set; }
```

Property Value

[bool](#)

SettleWaitTimeOut

Reads or sets the global default for settling time-out.

```
public double SettleWaitTimeOut { get; set; }
```

Property Value

[double](#)

VerboseMode

Reads or sets verbose mode (prints detailed information to log output target).

```
public bool VerboseMode { get; set; }
```

Property Value

[bool](#)

Methods

Add(Setup)

Adds the specified setup to the SetupService.

```
public void Add(Setup setup)
```

Parameters

setup [Setup](#)

The setup to add.

Apply(string)

Applies one or multiple named setups from the SetupService. If more than one is specified, the requested sequence is maintained. Does nothing if the name is null or empty.

```
public void Apply(string setupNames)
```

Parameters

setupNames [string](#) ↗

The setup(s) to be applied (CSV).

Diff(string)

Performs a diff of the specified setups against current hardware state and logs the differences to the output window.

```
public void Diff(string setupNames)
```

Parameters

setupNames [string](#) ↗

Dump()

Dumps all setups to the log output target.

```
public void Dump()
```

EnqueueExportData(string, bool)

```
public static void EnqueueExportData(string line, bool newLine = true)
```

Parameters

`line` [string](#)

`newLine` [bool](#)

Export(string, string)

Exports the specified setups to a file at the specified path. The file format is JSON. If the file already exists, it will be overwritten. If the path is null or empty, the setups will not be exported. An empty parameter `setupNames` will export all setups in the SetupService.

```
public void Export(string path, string setupNames)
```

Parameters

`path` [string](#)

The file system path where the setups will be exported.

`setupNames` [string](#)

The setup(s) to be exported (CSV).

Import(string, bool)

Imports setups from a file at the specified path. If overwrite is true, setups with the same name are overwritten.

```
public void Import(string path, bool overwrite = true)
```

Parameters

path [string](#)

The file system path from where the setups will be imported.

overwrite [bool](#)

Optional. If set to `true`, existing setups will be overwritten.

Init(InitMode)

Performs the specified initialization to all setups in the SetupService.

```
public void Init(InitMode initMode)
```

Parameters

initMode [InitMode](#)

The init mode.

Log(string, int, int)

Logs a message with a specified severity level and color.

```
public void Log(string message, int level, int rgb)
```

Parameters

message [string](#)

The message to be logged. Cannot be null or empty.

level [int](#)

The level of indentation for the message.

rgb [int](#)

The RGB color value used to display the message. Must be a valid RGB integer.

Remove(string)

Removes the setup with the specified setupName from the SetupService.

```
public bool Remove(string setupName)
```

Parameters

setupName [string](#)

The setup to be removed.

Returns

[bool](#)

[true](#) if the **setup** is successfully found and removed; otherwise, [false](#).

Reset()

Initialize the service. This is called by the API when the service is first used.

```
public void Reset()
```

Class Storage

Namespace: [Csra.Services](#)

Assembly: Csra.dll

StorageService - centralized persistent data storage.

```
public class Storage : IStorageService, IService
```

Inheritance

[object](#) ← Storage

Implements

[IStorageService](#), [IService](#)

Inherited Members

[object.ToString\(\)](#), [object.Equals\(object\)](#), [object.Equals\(object, object\)](#),
[object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#)

Properties

Count

Gets the number of key/value pairs contained in the storage.

```
public int Count { get; }
```

Property Value

[int](#)

Keys

Gets a collection containing the keys in the storage.

```
public IEnumerable<string> Keys { get; }
```

Property Value

[IEnumerable](#) <[string](#)>

Methods

AddOrUpdate(string, object)

Adds a key/value pair to the storage if the key does not already exist, or updates the value for an existing key.

```
public void AddOrUpdate(string key, object value)
```

Parameters

key [string](#)

The key to be added or whose value should be updated.

value [object](#)

The value of the element to add, or update.

ContainsKey(string)

Determines whether the storage contains the specified key.

```
public bool ContainsKey(string key)
```

Parameters

key [string](#)

The key to locate in the storage.

Returns

[bool](#)

[true](#) if the storage contains an element with the specified key; otherwise, [false](#).

Remove(string)

Removes the value with the specified key from the storage.

```
public bool Remove(string key)
```

Parameters

key [string](#)

The key of the element to remove.

Returns

[bool](#)

[true](#) if the element is successfully found and removed; otherwise [false](#).

Reset()

Initialize the service. This is called by the API when the service is first used.

```
public void Reset()
```

TryGetValue<T>(string, out T)

Gets the type-safe value associated with the specified key.

```
public bool TryGetValue<T>(string key, out T value)
```

Parameters

key [string](#)

The key of the value to get.

value T

When this methods returns, contains the value associated with the specified key, if the key is found; otherwise, the default value for the type of the value parameter. This parameter is passed uninitialized.

Returns

[bool](#)

[true](#) if the storage contains an element with the specified key; otherwise, [false](#).

Type Parameters

T

The type of object to retrieve.

Namespace Csra.Settings

Namespaces

[Csra.Settings.TheHdw](#)

Classes

[Custom<T>](#)

[SettingBase<T>](#)

Base class for all Services.Setup settings.

[Setting_Enum<T>](#)

[Setting_FlaggedEnum<T>](#)

[Setting_bool](#)

[Setting_double](#)

[Setting_int](#)

Namespace Csra.Settings.TheHdw

Namespaces

[Csra.Settings.TheHdw.Dcv.Pins](#)

[Csra.Settings.TheHdw.Dcv.Pins](#)

[Csra.Settings.TheHdw.Digital.Pins](#)

[Csra.Settings.TheHdw.Ppmu.Pins](#)

[Csra.Settings.TheHdw.Utility.Pins](#)

Classes

[SetSettlingTimer](#)

[SettleWait](#)

[Wait](#)

Namespace Csra.Setting.TheHdw.Dcv.Pins

Namespaces

[Csra.Setting.TheHdw.Dcv.Pins.BleederResistor](#)

[Csra.Setting.TheHdw.Dcv.Pins.FoldCurrentLimit](#)

Classes

[ComplianceRange_Negative](#)

[ComplianceRange_Positive](#)

[Connect](#)

[Current](#)

[CurrentRange](#)

[Gate](#)

[Mode](#)

[NominalBandwidth](#)

[Voltage](#)

[VoltageRange](#)

Namespace Csra.Settings.TheHdw.Dcv.Pins. BleederResistor

Classes

[CurrentLoad](#)

[Mode](#)

Class CurrentLoad

Namespace: [Csra.Settings.TheHdw.Dcv.Pins.BleederResistor](#)

Assembly: Csra.dll

```
public class CurrentLoad : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← CurrentLoad

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

CurrentLoad(double, string)

```
public CurrentLoad(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Mode

Namespace: [Csra.Settings.TheHdw.Dcv.Pins.BleederResistor](#)

Assembly: Csra.dll

```
public class Mode : Setting_Enum<tIDCVIBleederResistor>, ISetting
```

Inheritance

[object](#) ← [SettingBase](#)<tIDCVIBleederResistor> ← [Setting_Enum](#)<tIDCVIBleederResistor> ← Mode

Implements

[ISetting](#)

Inherited Members

[Setting_Enum<tIDCVIBleederResistor>.SerializeValue\(tIDCVIBleederResistor\)](#) ,
[SettingBase<tIDCVIBleederResistor>.pins](#) , [SettingBase<tIDCVIBleederResistor>.unit](#) ,
[SettingBase<tIDCVIBleederResistor>.SetArguments\(tIDCVIBleederResistor, string, bool\)](#) ,
[SettingBase<tIDCVIBleederResistor>.SetArguments\(tIDCVIBleederResistor, string\)](#) ,
[SettingBase<tIDCVIBleederResistor>.SetBehavior\(tIDCVIBleederResistor, string, InitMode, bool\)](#) ,
[SettingBase<tIDCVIBleederResistor>.SetContext\(Action<string, tIDCVIBleederResistor>, Func<string, tIDCVIBleederResistor\[\]>, Dictionary<string, tIDCVIBleederResistor>\)](#) ,
[SettingBase<tIDCVIBleederResistor>.CompareValue\(tIDCVIBleederResistor, tIDCVIBleederResistor\)](#) ,
[SettingBase<tIDCVIBleederResistor>.Validate\(\)](#) ,
[SettingBase<tIDCVIBleederResistor>.SetCacheInternal\(tIDCVIBleederResistor, string, Dictionary<string, tIDCVIBleederResistor>\)](#) ,
[SettingBase<tIDCVIBleederResistor>.Apply\(\)](#) , [SettingBase<tIDCVIBleederResistor>.Diff\(\)](#) ,
[SettingBase<tIDCVIBleederResistor>.ToString\(\)](#) , [SettingBase<tIDCVIBleederResistor>.Init\(InitMode\)](#) ,
[SettingBase<tIDCVIBleederResistor>.Dump\(\)](#) , [SettingBase<tIDCVIBleederResistor>.Export\(string\)](#) ,
[SettingBase<tIDCVIBleederResistor>.Indent\(int\)](#) ,
[SettingBase<tIDCVIBleederResistor>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Mode(tIDCVIBleederResistor, string)

```
public Mode(tlDCVIBleederResistor value, string pinList)
```

Parameters

value tlDCVIBleederResistor

pinList [string](#)

Methods

SetCache(tlDCVIBleederResistor, string)

```
public static void SetCache(tlDCVIBleederResistor value, string pinList)
```

Parameters

value tlDCVIBleederResistor

pinList [string](#)

Namespace Csra.Settings.TheHdw.Dcv.Pins.Fold CurrentLimit

Classes

[Behavior](#)

[Timeout](#)

Class Behavior

Namespace: [Csra.Settings.TheHdw.Dcv.Pins.FoldCurrentLimit](#)

Assembly: Csra.dll

```
public class Behavior : Setting_Enum<tlDCVIFoldCurrentLimitBehavior>, ISetting
```

Inheritance

[object](#) ← [SettingBase](#)<tlDCVIFoldCurrentLimitBehavior> ←
[Setting_Enum](#)<tlDCVIFoldCurrentLimitBehavior> ← Behavior

Implements

[ISetting](#)

Inherited Members

[Setting_Enum<tlDCVIFoldCurrentLimitBehavior>.SerializeValue\(tlDCVIFoldCurrentLimitBehavior\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.pins](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.unit](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.SetArguments\(tlDCVIFoldCurrentLimitBehavior, string, bool\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.SetArguments\(tlDCVIFoldCurrentLimitBehavior, string\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.SetBehavior\(tlDCVIFoldCurrentLimitBehavior, string, InitMode, bool\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.SetContext\(Action<string, tlDCVIFoldCurrentLimitBehavior>, Func<string, tlDCVIFoldCurrentLimitBehavior\[\]>, Dictionary<string, tlDCVIFoldCurrentLimitBehavior>\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.CompareValue\(tlDCVIFoldCurrentLimitBehavior, tlDCVIFoldCurrentLimitBehavior\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.Validate\(\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.SetCacheInternal\(tlDCVIFoldCurrentLimitBehavior, string, Dictionary<string, tlDCVIFoldCurrentLimitBehavior>\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.Apply\(\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.Diff\(\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.ToString\(\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.Init\(InitMode\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.Dump\(\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.Export\(string\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.Indent\(int\)](#),
[SettingBase<tlDCVIFoldCurrentLimitBehavior>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),

[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Behavior(tlDCVIFoldCurrentLimitBehavior, string)

```
public Behavior(tlDCVIFoldCurrentLimitBehavior value, string pinList)
```

Parameters

value tlDCVIFoldCurrentLimitBehavior

pinList [string](#)

Methods

SetCache(tlDCVIFoldCurrentLimitBehavior, string)

```
public static void SetCache(tlDCVIFoldCurrentLimitBehavior value, string pinList)
```

Parameters

value tlDCVIFoldCurrentLimitBehavior

pinList [string](#)

Class Timeout

Namespace: [Csra.Settings.TheHdw.Dcv.Pins.FoldCurrentLimit](#)

Assembly: Csra.dll

```
public class Timeout : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Timeout

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#) , [SettingBase<double>.pins](#) , [SettingBase<double>.unit](#) ,
[SettingBase<double>.SetArguments\(double, string, bool\)](#) ,
[SettingBase<double>.SetArguments\(double, string\)](#) ,
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#) ,
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#) ,
[SettingBase<double>.CompareValue\(double, double\)](#) , [SettingBase<double>.Validate\(\)](#) ,
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#) ,
[SettingBase<double>.Apply\(\)](#) , [SettingBase<double>.Diff\(\)](#) , [SettingBase<double>.ToString\(\)](#) ,
[SettingBase<double>.Init\(InitMode\)](#) , [SettingBase<double>.Dump\(\)](#) ,
[SettingBase<double>.Export\(string\)](#) , [SettingBase<double>.Indent\(int\)](#) ,
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Timeout(double, string)

```
public Timeout(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class ComplianceRange_Negative

Namespace: [Csra.Settings.TheHdw.Dcv.Pins](#)

Assembly: Csra.dll

```
public class ComplianceRange_Negative : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← ComplianceRange_Negative

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

ComplianceRange_Negative(double, string)

```
public ComplianceRange_Negative(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class ComplianceRange_Positive

Namespace: [Csra.Settings.TheHdw.Dcv.Pins](#)

Assembly: Csra.dll

```
public class ComplianceRange_Positive : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← ComplianceRange_Positive

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

ComplianceRange_Positive(double, string)

```
public ComplianceRange_Positive(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Connect

Namespace: [Csra.Settings.TheHdw.Dcv.Pins](#)

Assembly: Csra.dll

```
public class Connect : Setting_Enum<tIDCVIConnectWhat>, ISetting
```

Inheritance

[object](#) ← [SettingBase](#)<tIDCVIConnectWhat> ← [Setting_Enum](#)<tIDCVIConnectWhat> ← Connect

Implements

[ISetting](#)

Inherited Members

[Setting_Enum](#)<tIDCVIConnectWhat>.SerializeValue(tIDCVIConnectWhat),
[SettingBase](#)<tIDCVIConnectWhat>.pins, [SettingBase](#)<tIDCVIConnectWhat>.unit,
[SettingBase](#)<tIDCVIConnectWhat>.SetArguments(tIDCVIConnectWhat, string, bool),
[SettingBase](#)<tIDCVIConnectWhat>.SetArguments(tIDCVIConnectWhat, string),
[SettingBase](#)<tIDCVIConnectWhat>.SetBehavior(tIDCVIConnectWhat, string, InitMode, bool),
[SettingBase](#)<tIDCVIConnectWhat>.SetContext(Action<string, tIDCVIConnectWhat>, Func<string, tIDCVIConnectWhat[]>, Dictionary<string, tIDCVIConnectWhat>),
[SettingBase](#)<tIDCVIConnectWhat>.CompareValue(tIDCVIConnectWhat, tIDCVIConnectWhat),
[SettingBase](#)<tIDCVIConnectWhat>.Validate(),
[SettingBase](#)<tIDCVIConnectWhat>.SetCacheInternal(tIDCVIConnectWhat, string, Dictionary<string, tIDCVIConnectWhat>),
[SettingBase](#)<tIDCVIConnectWhat>.Apply(), [SettingBase](#)<tIDCVIConnectWhat>.Diff(),
[SettingBase](#)<tIDCVIConnectWhat>.ToString(), [SettingBase](#)<tIDCVIConnectWhat>.Init(InitMode),
[SettingBase](#)<tIDCVIConnectWhat>.Dump(), [SettingBase](#)<tIDCVIConnectWhat>.Export(string),
[SettingBase](#)<tIDCVIConnectWhat>.Indent(int),
[SettingBase](#)<tIDCVIConnectWhat>.ParseEnum<TEnum>(string), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

Connect(tIDCVIConnectWhat, string)

```
public Connect(tlDCVIConnectWhat value, string pinList)
```

Parameters

value tlDCVIConnectWhat

pinList [string](#)

Methods

SetCache(tlDCVIConnectWhat, string)

```
public static void SetCache(tlDCVIConnectWhat value, string pinList)
```

Parameters

value tlDCVIConnectWhat

pinList [string](#)

Class Current

Namespace: [Csra.Settings.TheHdw.Dcv.Pins](#)

Assembly: Csra.dll

```
public class Current : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Current

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#) , [SettingBase<double>.pins](#) , [SettingBase<double>.unit](#) ,
[SettingBase<double>.SetArguments\(double, string, bool\)](#) ,
[SettingBase<double>.SetArguments\(double, string\)](#) ,
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#) ,
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#) ,
[SettingBase<double>.CompareValue\(double, double\)](#) , [SettingBase<double>.Validate\(\)](#) ,
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#) ,
[SettingBase<double>.Apply\(\)](#) , [SettingBase<double>.Diff\(\)](#) , [SettingBase<double>.ToString\(\)](#) ,
[SettingBase<double>.Init\(InitMode\)](#) , [SettingBase<double>.Dump\(\)](#) ,
[SettingBase<double>.Export\(string\)](#) , [SettingBase<double>.Indent\(int\)](#) ,
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Current(double, string)

```
public Current(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class CurrentRange

Namespace: [Csra.Settings.TheHdw.Dcv.Pins](#)

Assembly: Csra.dll

```
public class CurrentRange : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← CurrentRange

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#) , [SettingBase<double>.pins](#) , [SettingBase<double>.unit](#) ,
[SettingBase<double>.SetArguments\(double, string, bool\)](#) ,
[SettingBase<double>.SetArguments\(double, string\)](#) ,
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#) ,
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#) ,
[SettingBase<double>.CompareValue\(double, double\)](#) , [SettingBase<double>.Validate\(\)](#) ,
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#) ,
[SettingBase<double>.Apply\(\)](#) , [SettingBase<double>.Diff\(\)](#) , [SettingBase<double>.ToString\(\)](#) ,
[SettingBase<double>.Init\(InitMode\)](#) , [SettingBase<double>.Dump\(\)](#) ,
[SettingBase<double>.Export\(string\)](#) , [SettingBase<double>.Indent\(int\)](#) ,
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

CurrentRange(double, string)

```
public CurrentRange(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Gate

Namespace: [Csra.Settings.TheHdw.Dcv.Pins](#)

Assembly: Csra.dll

```
public class Gate : Setting_Enum<tIDCVGate>, ISetting
```

Inheritance

[object](#) ← [SettingBase](#)<tIDCVGate> ← [Setting_Enum](#)<tIDCVGate> ← Gate

Implements

[ISetting](#)

Inherited Members

[Setting_Enum<tIDCVGate>.SerializeValue\(tIDCVGate\)](#), [SettingBase<tIDCVGate>.pins](#),
[SettingBase<tIDCVGate>.unit](#), [SettingBase<tIDCVGate>.SetArguments\(tIDCVGate, string, bool\)](#),
[SettingBase<tIDCVGate>.SetArguments\(tIDCVGate, string\)](#),
[SettingBase<tIDCVGate>.SetBehavior\(tIDCVGate, string, InitMode, bool\)](#),
[SettingBase<tIDCVGate>.SetContext\(Action<string, tIDCVGate>, Func<string, tIDCVGate\[\]>, Dictionary<string, tIDCVGate>\)](#),
[SettingBase<tIDCVGate>.CompareValue\(tIDCVGate, tIDCVGate\)](#), [SettingBase<tIDCVGate>.Validate\(\)](#),
[SettingBase<tIDCVGate>.SetCacheInternal\(tIDCVGate, string, Dictionary<string, tIDCVGate>\)](#),
[SettingBase<tIDCVGate>.Apply\(\)](#), [SettingBase<tIDCVGate>.Diff\(\)](#), [SettingBase<tIDCVGate>.ToString\(\)](#),
[SettingBase<tIDCVGate>.Init\(InitMode\)](#), [SettingBase<tIDCVGate>.Dump\(\)](#),
[SettingBase<tIDCVGate>.Export\(string\)](#), [SettingBase<tIDCVGate>.Indent\(int\)](#),
[SettingBase<tIDCVGate>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

Gate(tIDCVGate, string)

```
public Gate(tIDCVGate value, string pinList)
```

Parameters

value tlDCVGate

pinList string ↗

Methods

SetCache(tlDCVGate, string)

```
public static void SetCache(tlDCVGate value, string pinList)
```

Parameters

value tlDCVGate

pinList string ↗

Class Mode

Namespace: [Csra.Settings.TheHdw.Dcv.Pins](#)

Assembly: Csra.dll

```
public class Mode : Setting_Enum<tIDCVIMode>, ISetting
```

Inheritance

[object](#) ← [SettingBase](#)<tIDCVIMode> ← [Setting_Enum](#)<tIDCVIMode> ← Mode

Implements

[ISetting](#)

Inherited Members

[Setting_Enum<tIDCVIMode>.SerializeValue\(tIDCVIMode\)](#), [SettingBase<tIDCVIMode>.pins](#),
[SettingBase<tIDCVIMode>.unit](#), [SettingBase<tIDCVIMode>.SetArguments\(tIDCVIMode, string, bool\)](#),
[SettingBase<tIDCVIMode>.SetArguments\(tIDCVIMode, string\)](#),
[SettingBase<tIDCVIMode>.SetBehavior\(tIDCVIMode, string, InitMode, bool\)](#),
[SettingBase<tIDCVIMode>.SetContext\(Action<string, tIDCVIMode>, Func<string, tIDCVIMode\[\]>, Dictionary<string, tIDCVIMode>\)](#),
[SettingBase<tIDCVIMode>.CompareValue\(tIDCVIMode, tIDCVIMode\)](#),
[SettingBase<tIDCVIMode>.Validate\(\)](#),
[SettingBase<tIDCVIMode>.SetCacheInternal\(tIDCVIMode, string, Dictionary<string, tIDCVIMode>\)](#),
[SettingBase<tIDCVIMode>.Apply\(\)](#), [SettingBase<tIDCVIMode>.Diff\(\)](#),
[SettingBase<tIDCVIMode>.ToString\(\)](#), [SettingBase<tIDCVIMode>.Init\(InitMode\)](#),
[SettingBase<tIDCVIMode>.Dump\(\)](#), [SettingBase<tIDCVIMode>.Export\(string\)](#),
[SettingBase<tIDCVIMode>.Indent\(int\)](#), [SettingBase<tIDCVIMode>.ParseEnum<TEnum>\(string\)](#),
[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#),
[object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

Mode(tIDCVIMode, string)

```
public Mode(tIDCVIMode value, string pinList)
```

Parameters

value tIDCVIMode

pinList string☒

Methods

SetCache(tIDCVIMode, string)

```
public static void SetCache(tIDCVIMode value, string pinList)
```

Parameters

value tIDCVIMode

pinList string☒

Class NominalBandwidth

Namespace: [Csra.Settings.TheHdw.Dcv.Pins](#)

Assembly: Csra.dll

```
public class NominalBandwidth : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← NominalBandwidth

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

NominalBandwidth(double, string)

```
public NominalBandwidth(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Voltage

Namespace: [Csra.Settings.TheHdw.Dcv.Pins](#)

Assembly: Csra.dll

```
public class Voltage : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Voltage

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#) , [SettingBase<double>.pins](#) , [SettingBase<double>.unit](#) ,
[SettingBase<double>.SetArguments\(double, string, bool\)](#) ,
[SettingBase<double>.SetArguments\(double, string\)](#) ,
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#) ,
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#) ,
[SettingBase<double>.CompareValue\(double, double\)](#) , [SettingBase<double>.Validate\(\)](#) ,
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#) ,
[SettingBase<double>.Apply\(\)](#) , [SettingBase<double>.Diff\(\)](#) , [SettingBase<double>.ToString\(\)](#) ,
[SettingBase<double>.Init\(InitMode\)](#) , [SettingBase<double>.Dump\(\)](#) ,
[SettingBase<double>.Export\(string\)](#) , [SettingBase<double>.Indent\(int\)](#) ,
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Voltage(double, string)

```
public Voltage(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class VoltageRange

Namespace: [Csra.Settings.TheHdw.Dcv.Pins](#)

Assembly: Csra.dll

```
public class VoltageRange : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← VoltageRange

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

VoltageRange(double, string)

```
public VoltageRange(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Namespace Csra.Settings.TheHdw.Dcvs.Pins

Namespaces

[Csra.Settings.TheHdw.Dcvs.Pins.CurrentLimit](#)

Classes

[BleederResistor](#)

[Connect](#)

[CurrentRange](#)

[Gate](#)

[Mode](#)

[Voltage](#)

[VoltageRange](#)

Namespace Csra.Settings.TheHdw.Dcvs.Pins. CurrentLimit

Namespaces

[Csra.Settings.TheHdw.Dcvs.Pins.CurrentLimit.Sink](#)

[Csra.Settings.TheHdw.Dcvs.Pins.CurrentLimit.Source](#)

Namespace Csra.Settings.TheHdw.Dcvs.Pins.CurrentLimit.Sink

Namespaces

[Csra.Settings.TheHdw.Dcvs.Pins.CurrentLimit.Sink.FoldLimit](#)

[Csra.Settings.TheHdw.Dcvs.Pins.CurrentLimit.Sink.OverloadLimit](#)

Namespace Csra.Settings.TheHdw.Dcvs.Pins. CurrentLimit.Sink.FoldLimit

Classes

[Level](#)

Class Level

Namespace: [Csra.Settings.TheHdw.Dcvs.Pins.CurrentLimit.Sink.FoldLimit](#)

Assembly: Csra.dll

```
public class Level : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Level

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

Level(double, string)

```
public Level(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Namespace Csra.Settings.TheHdw.Dcvs.Pins. CurrentLimit.Sink.OverloadLimit

Classes

[Level](#)

Class Level

Namespace: [Csra.Settings.TheHdw.Dcvs.Pins.CurrentLimit.Sink.OverloadLimit](#)

Assembly: Csra.dll

```
public class Level : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Level

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#) , [SettingBase<double>.pins](#) , [SettingBase<double>.unit](#) ,
[SettingBase<double>.SetArguments\(double, string, bool\)](#) ,
[SettingBase<double>.SetArguments\(double, string\)](#) ,
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#) ,
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#) ,
[SettingBase<double>.CompareValue\(double, double\)](#) , [SettingBase<double>.Validate\(\)](#) ,
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#) ,
[SettingBase<double>.Apply\(\)](#) , [SettingBase<double>.Diff\(\)](#) , [SettingBase<double>.ToString\(\)](#) ,
[SettingBase<double>.Init\(InitMode\)](#) , [SettingBase<double>.Dump\(\)](#) ,
[SettingBase<double>.Export\(string\)](#) , [SettingBase<double>.Indent\(int\)](#) ,
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Level(double, string)

```
public Level(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Namespace Csra.Settings.TheHdw.Dcvs.Pins.CurrentLimit.Source

Namespaces

[Csra.Settings.TheHdw.Dcvs.Pins.CurrentLimit.Source.FoldLimit](#)

[Csra.Settings.TheHdw.Dcvs.Pins.CurrentLimit.Source.OverloadLimit](#)

Namespace Csra.Settings.TheHdw.Dcvs.Pins. CurrentLimit.Source.FoldLimit

Classes

[Level](#)

Class Level

Namespace: [Csra.Settings.TheHdw.Dcvs.Pins.CurrentLimit.Source.FoldLimit](#)

Assembly: Csra.dll

```
public class Level : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Level

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#) , [SettingBase<double>.pins](#) , [SettingBase<double>.unit](#) ,
[SettingBase<double>.SetArguments\(double, string, bool\)](#) ,
[SettingBase<double>.SetArguments\(double, string\)](#) ,
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#) ,
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#) ,
[SettingBase<double>.CompareValue\(double, double\)](#) , [SettingBase<double>.Validate\(\)](#) ,
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#) ,
[SettingBase<double>.Apply\(\)](#) , [SettingBase<double>.Diff\(\)](#) , [SettingBase<double>.ToString\(\)](#) ,
[SettingBase<double>.Init\(InitMode\)](#) , [SettingBase<double>.Dump\(\)](#) ,
[SettingBase<double>.Export\(string\)](#) , [SettingBase<double>.Indent\(int\)](#) ,
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Level(double, string)

```
public Level(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Namespace Csra.Settings.TheHdw.Dcvs.Pins. CurrentLimit.Source.OverloadLimit

Classes

[Level](#)

Class Level

Namespace: [Csra.Settings.TheHdw.Dcvs.Pins.CurrentLimit.Source.OverloadLimit](#)

Assembly: Csra.dll

```
public class Level : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Level

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#) , [SettingBase<double>.pins](#) , [SettingBase<double>.unit](#) ,
[SettingBase<double>.SetArguments\(double, string, bool\)](#) ,
[SettingBase<double>.SetArguments\(double, string\)](#) ,
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#) ,
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#) ,
[SettingBase<double>.CompareValue\(double, double\)](#) , [SettingBase<double>.Validate\(\)](#) ,
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#) ,
[SettingBase<double>.Apply\(\)](#) , [SettingBase<double>.Diff\(\)](#) , [SettingBase<double>.ToString\(\)](#) ,
[SettingBase<double>.Init\(InitMode\)](#) , [SettingBase<double>.Dump\(\)](#) ,
[SettingBase<double>.Export\(string\)](#) , [SettingBase<double>.Indent\(int\)](#) ,
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Level(double, string)

```
public Level(double value, string pinList)
```

Parameters

value double ↴

pinList string ↴

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value double ↴

pinList string ↴

Class BleederResistor

Namespace: [Csra.Settings.TheHdw.Dcvs.Pins](#)

Assembly: Csra.dll

```
public class BleederResistor : Setting_Enum<tlDCVSOnOffAuto>, ISetting
```

Inheritance

[object](#) ← [SettingBase](#)<tlDCVSOnOffAuto> ← [Setting_Enum](#)<tlDCVSOnOffAuto> ← BleederResistor

Implements

[ISetting](#)

Inherited Members

[Setting_Enum<tlDCVSOnOffAuto>.SerializeValue\(tlDCVSOnOffAuto\)](#),
[SettingBase<tlDCVSOnOffAuto>.pins](#), [SettingBase<tlDCVSOnOffAuto>.unit](#),
[SettingBase<tlDCVSOnOffAuto>.SetArguments\(tlDCVSOnOffAuto, string, bool\)](#),
[SettingBase<tlDCVSOnOffAuto>.SetArguments\(tlDCVSOnOffAuto, string\)](#),
[SettingBase<tlDCVSOnOffAuto>.SetBehavior\(tlDCVSOnOffAuto, string, InitMode, bool\)](#),
[SettingBase<tlDCVSOnOffAuto>.SetContext\(Action<string, tlDCVSOnOffAuto>, Func<string, tlDCVSOnOffAuto\[\]>, Dictionary<string, tlDCVSOnOffAuto>\)](#),
[SettingBase<tlDCVSOnOffAuto>.CompareValue\(tlDCVSOnOffAuto, tlDCVSOnOffAuto\)](#),
[SettingBase<tlDCVSOnOffAuto>.Validate\(\)](#),
[SettingBase<tlDCVSOnOffAuto>.SetCacheInternal\(tlDCVSOnOffAuto, string, Dictionary<string, tlDCVSOnOffAuto>\)](#),
[SettingBase<tlDCVSOnOffAuto>.Apply\(\)](#), [SettingBase<tlDCVSOnOffAuto>.Diff\(\)](#),
[SettingBase<tlDCVSOnOffAuto>.ToString\(\)](#), [SettingBase<tlDCVSOnOffAuto>.Init\(InitMode\)](#),
[SettingBase<tlDCVSOnOffAuto>.Dump\(\)](#), [SettingBase<tlDCVSOnOffAuto>.Export\(string\)](#),
[SettingBase<tlDCVSOnOffAuto>.Indent\(int\)](#),
[SettingBase<tlDCVSOnOffAuto>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

BleederResistor(tlDCVSOnOffAuto, string)

```
public BleederResistor(tlDCVSOnOffAuto value, string pinList)
```

Parameters

value tlDCVSOnOffAuto

pinList [string](#)

Methods

SetCache(tlDCVSOnOffAuto, string)

```
public static void SetCache(tlDCVSOnOffAuto value, string pinList)
```

Parameters

value tlDCVSOnOffAuto

pinList [string](#)

Class Connect

Namespace: [Csra.Settings.TheHdw.Dcvs.Pins](#)

Assembly: Csra.dll

```
public class Connect : Setting_Enum<tIDCVSConnectWhat>, ISetting
```

Inheritance

[object](#) ← [SettingBase](#)<tIDCVSConnectWhat> ← [Setting_Enum](#)<tIDCVSConnectWhat> ← Connect

Implements

[ISetting](#)

Inherited Members

[Setting_Enum<tIDCVSConnectWhat>.SerializeValue\(tIDCVSConnectWhat\)](#) ,
[SettingBase<tIDCVSConnectWhat>.pins](#) , [SettingBase<tIDCVSConnectWhat>.unit](#) ,
[SettingBase<tIDCVSConnectWhat>.SetArguments\(tIDCVSConnectWhat, string, bool\)](#) ,
[SettingBase<tIDCVSConnectWhat>.SetArguments\(tIDCVSConnectWhat, string\)](#) ,
[SettingBase<tIDCVSConnectWhat>.SetBehavior\(tIDCVSConnectWhat, string, InitMode, bool\)](#) ,
[SettingBase<tIDCVSConnectWhat>.SetContext\(Action<string, tIDCVSConnectWhat>, Func<string, tIDCVSConnectWhat\[\]>, Dictionary<string, tIDCVSConnectWhat>\)](#) ,
[SettingBase<tIDCVSConnectWhat>.CompareValue\(tIDCVSConnectWhat, tIDCVSConnectWhat\)](#) ,
[SettingBase<tIDCVSConnectWhat>.Validate\(\)](#) ,
[SettingBase<tIDCVSConnectWhat>.SetCacheInternal\(tIDCVSConnectWhat, string, Dictionary<string, tIDCVSConnectWhat>\)](#) ,
[SettingBase<tIDCVSConnectWhat>.Apply\(\)](#) , [SettingBase<tIDCVSConnectWhat>.Diff\(\)](#) ,
[SettingBase<tIDCVSConnectWhat>.ToString\(\)](#) , [SettingBase<tIDCVSConnectWhat>.Init\(InitMode\)](#) ,
[SettingBase<tIDCVSConnectWhat>.Dump\(\)](#) , [SettingBase<tIDCVSConnectWhat>.Export\(string\)](#) ,
[SettingBase<tIDCVSConnectWhat>.Indent\(int\)](#) ,
[SettingBase<tIDCVSConnectWhat>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Connect(tIDCVSConnectWhat, string)

```
public Connect(tlDCVSConnectWhat value, string pinList)
```

Parameters

value tlDCVSConnectWhat

pinList [string](#)

Methods

SetCache(tlDCVSConnectWhat, string)

```
public static void SetCache(tlDCVSConnectWhat value, string pinList)
```

Parameters

value tlDCVSConnectWhat

pinList [string](#)

Class CurrentRange

Namespace: [Csra.Settings.TheHdw.Dcvs.Pins](#)

Assembly: Csra.dll

```
public class CurrentRange : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← CurrentRange

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

CurrentRange(double, string)

```
public CurrentRange(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Gate

Namespace: [Csra.Settings.TheHdw.Dcvs.Pins](#)

Assembly: Csra.dll

```
public class Gate : Setting_bool, ISetting
```

Inheritance

[object](#) ← [SettingBase<bool>](#) ← [Setting_bool](#) ← Gate

Implements

[ISetting](#)

Inherited Members

[SettingBase<bool>.pins](#) , [SettingBase<bool>.unit](#) ,
[SettingBase<bool>.SetArguments\(bool, string, bool\)](#) , [SettingBase<bool>.SetArguments\(bool, string\)](#) ,
[SettingBase<bool>.SetBehavior\(bool, string, InitMode, bool\)](#) ,
[SettingBase<bool>.SetContext\(Action<string, bool>, Func<string, bool\[\]>, Dictionary<string, bool>\)](#) ,
[SettingBase<bool>.CompareValue\(bool, bool\)](#) , [SettingBase<bool>.Validate\(\)](#) ,
[SettingBase<bool>.SerializeValue\(bool\)](#) ,
[SettingBase<bool>.SetCacheInternal\(bool, string, Dictionary<string, bool>\)](#) ,
[SettingBase<bool>.Apply\(\)](#) , [SettingBase<bool>.Diff\(\)](#) , [SettingBase<bool>.ToString\(\)](#) ,
[SettingBase<bool>.Init\(InitMode\)](#) , [SettingBase<bool>.Dump\(\)](#) , [SettingBase<bool>.Export\(string\)](#) ,
[SettingBase<bool>.Indent\(int\)](#) , [SettingBase<bool>.ParseEnum<TEnum>\(string\)](#) ,
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) ,
[object.GetHashCode\(\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Gate(bool, string)

```
public Gate(bool value, string pinList)
```

Parameters

value [bool](#)

pinList [string](#)

Methods

SetCache(bool, string)

```
public static void SetCache(bool value, string pinList)
```

Parameters

value [bool](#)

pinList [string](#)

Class Mode

Namespace: [Csra.Settings.TheHdw.Dcvs.Pins](#)

Assembly: Csra.dll

```
public class Mode : Setting_Enum<tIDCVSMode>, ISetting
```

Inheritance

[object](#) ← [SettingBase](#)<tIDCVSMode> ← [Setting_Enum](#)<tIDCVSMode> ← Mode

Implements

[ISetting](#)

Inherited Members

[Setting_Enum<tIDCVSMode>.SerializeValue\(tIDCVSMode\)](#), [SettingBase<tIDCVSMode>.pins](#),
[SettingBase<tIDCVSMode>.unit](#), [SettingBase<tIDCVSMode>.SetArguments\(tIDCVSMode, string, bool\)](#),
[SettingBase<tIDCVSMode>.SetArguments\(tIDCVSMode, string\)](#),
[SettingBase<tIDCVSMode>.SetBehavior\(tIDCVSMode, string, InitMode, bool\)](#),
[SettingBase<tIDCVSMode>.SetContext\(Action<string, tIDCVSMode>, Func<string, tIDCVSMode\[\]>, Dictionary<string, tIDCVSMode>\)](#),
[SettingBase<tIDCVSMode>.CompareValue\(tIDCVSMode, tIDCVSMode\)](#),
[SettingBase<tIDCVSMode>.Validate\(\)](#),
[SettingBase<tIDCVSMode>.SetCacheInternal\(tIDCVSMode, string, Dictionary<string, tIDCVSMode>\)](#),
[SettingBase<tIDCVSMode>.Apply\(\)](#), [SettingBase<tIDCVSMode>.Diff\(\)](#),
[SettingBase<tIDCVSMode>.ToString\(\)](#), [SettingBase<tIDCVSMode>.Init\(InitMode\)](#),
[SettingBase<tIDCVSMode>.Dump\(\)](#), [SettingBase<tIDCVSMode>.Export\(string\)](#),
[SettingBase<tIDCVSMode>.Indent\(int\)](#), [SettingBase<tIDCVSMode>.ParseEnum<TEnum>\(string\)](#),
[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#),
[object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

Mode(tIDCVSMode, string)

```
public Mode(tIDCVSMode value, string pinList)
```

Parameters

value tIDCVSMode

pinList string ↗

Methods

SetCache(tIDCVSMode, string)

```
public static void SetCache(tIDCVSMode value, string pinList)
```

Parameters

value tIDCVSMode

pinList string ↗

Class Voltage

Namespace: [Csra.Settings.TheHdw.Dcvs.Pins](#)

Assembly: Csra.dll

```
public class Voltage : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Voltage

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#) , [SettingBase<double>.pins](#) , [SettingBase<double>.unit](#) ,
[SettingBase<double>.SetArguments\(double, string, bool\)](#) ,
[SettingBase<double>.SetArguments\(double, string\)](#) ,
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#) ,
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#) ,
[SettingBase<double>.CompareValue\(double, double\)](#) , [SettingBase<double>.Validate\(\)](#) ,
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#) ,
[SettingBase<double>.Apply\(\)](#) , [SettingBase<double>.Diff\(\)](#) , [SettingBase<double>.ToString\(\)](#) ,
[SettingBase<double>.Init\(InitMode\)](#) , [SettingBase<double>.Dump\(\)](#) ,
[SettingBase<double>.Export\(string\)](#) , [SettingBase<double>.Indent\(int\)](#) ,
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Voltage(double, string)

```
public Voltage(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class VoltageRange

Namespace: [Csra.Settings.TheHdw.Dcvs.Pins](#)

Assembly: Csra.dll

```
public class VoltageRange : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← VoltageRange

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

VoltageRange(double, string)

```
public VoltageRange(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Namespace Csra.Settings.TheHdw.Digital.Pins

Namespaces

[Csra.Settings.TheHdw.Digital.Pins.Levels](#)

Classes

[Connect](#)

[InitState](#)

[StartState](#)

Namespace Csra.Settings.TheHdw.Digital.Pins.Levels

Classes

[DriverMode](#)

[Value_Ioh](#)

[Value_Iol](#)

[Value_Vch](#)

[Value_Vcl](#)

[Value_Vih](#)

[Value_Vil](#)

[Value_Voh](#)

[Value_Vol](#)

[Value_Vt](#)

Class DriverMode

Namespace: [Csra.Settings.TheHdw.Digital.Pins.Levels](#)

Assembly: Csra.dll

```
public class DriverMode : Setting_Enum<tlDriverMode>, ISetting
```

Inheritance

[object](#) ← [SettingBase](#)<tlDriverMode> ← [Setting_Enum](#)<tlDriverMode> ← DriverMode

Implements

[ISetting](#)

Inherited Members

[Setting_Enum<tlDriverMode>.SerializeValue\(tlDriverMode\)](#), [SettingBase<tlDriverMode>.pins](#),
[SettingBase<tlDriverMode>.unit](#),
[SettingBase<tlDriverMode>.SetArguments\(tlDriverMode, string, bool\)](#),
[SettingBase<tlDriverMode>.SetArguments\(tlDriverMode, string\)](#),
[SettingBase<tlDriverMode>.SetBehavior\(tlDriverMode, string, InitMode, bool\)](#),
[SettingBase<tlDriverMode>.SetContext\(Action<string, tlDriverMode>, Func<string, tlDriverMode\[\]>, Dictionary<string, tlDriverMode>\)](#),
[SettingBase<tlDriverMode>.CompareValue\(tlDriverMode, tlDriverMode\)](#),
[SettingBase<tlDriverMode>.Validate\(\)](#),
[SettingBase<tlDriverMode>.SetCacheInternal\(tlDriverMode, string, Dictionary<string, tlDriverMode>\)](#),
[SettingBase<tlDriverMode>.Apply\(\)](#), [SettingBase<tlDriverMode>.Diff\(\)](#),
[SettingBase<tlDriverMode>.ToString\(\)](#), [SettingBase<tlDriverMode>.Init\(InitMode\)](#),
[SettingBase<tlDriverMode>.Dump\(\)](#), [SettingBase<tlDriverMode>.Export\(string\)](#),
[SettingBase<tlDriverMode>.Indent\(int\)](#), [SettingBase<tlDriverMode>.ParseEnum<TEnum>\(string\)](#),
[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#),
[object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

DriverMode(tlDriverMode, string)

```
public DriverMode(tlDriverMode value, string pinList)
```

Parameters

value tlDriverMode

pinList [string](#)

Methods

SetCache(tlDriverMode, string)

```
public static void SetCache(tlDriverMode value, string pinList)
```

Parameters

value tlDriverMode

pinList [string](#)

Class Value_Ioh

Namespace: [Csra.Settings.TheHdw.Digital.Pins.Levels](#)

Assembly: Csra.dll

```
public class Value_Ioh : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Value_Ioh

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#) , [SettingBase<double>.pins](#) , [SettingBase<double>.unit](#) ,
[SettingBase<double>.SetArguments\(double, string, bool\)](#) ,
[SettingBase<double>.SetArguments\(double, string\)](#) ,
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#) ,
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#) ,
[SettingBase<double>.CompareValue\(double, double\)](#) , [SettingBase<double>.Validate\(\)](#) ,
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#) ,
[SettingBase<double>.Apply\(\)](#) , [SettingBase<double>.Diff\(\)](#) , [SettingBase<double>.ToString\(\)](#) ,
[SettingBase<double>.Init\(InitMode\)](#) , [SettingBase<double>.Dump\(\)](#) ,
[SettingBase<double>.Export\(string\)](#) , [SettingBase<double>.Indent\(int\)](#) ,
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Value_Ioh(double, string)

```
public Value_Ioh(double value, string pinList)
```

Parameters

value double ↴

pinList string ↴

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value double ↴

pinList string ↴

Class Value_Iol

Namespace: [Csra.Settings.TheHdw.Digital.Pins.Levels](#)

Assembly: Csra.dll

```
public class Value_Iol : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Value_Iol

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

Value_Iol(double, string)

```
public Value_Iol(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Value_Vch

Namespace: [Csra.Settings.TheHdw.Digital.Pins.Levels](#)

Assembly: Csra.dll

```
public class Value_Vch : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Value_Vch

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

Value_Vch(double, string)

```
public Value_Vch(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Value_Vcl

Namespace: [Csra.Settings.TheHdw.Digital.Pins.Levels](#)

Assembly: Csra.dll

```
public class Value_Vcl : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Value_Vcl

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

Value_Vcl(double, string)

```
public Value_Vcl(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Value_Vih

Namespace: [Csra.Settings.TheHdw.Digital.Pins.Levels](#)

Assembly: Csra.dll

```
public class Value_Vih : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Value_Vih

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

Value_Vih(double, string)

```
public Value_Vih(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Value_Vil

Namespace: [Csra.Settings.TheHdw.Digital.Pins.Levels](#)

Assembly: Csra.dll

```
public class Value_Vil : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Value_Vil

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

Value_Vil(double, string)

```
public Value_Vil(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Value_Voh

Namespace: [Csra.Settings.TheHdw.Digital.Pins.Levels](#)

Assembly: Csra.dll

```
public class Value_Voh : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Value_Voh

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

Value_Voh(double, string)

```
public Value_Voh(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Value_Vol

Namespace: [Csra.Settings.TheHdw.Digital.Pins.Levels](#)

Assembly: Csra.dll

```
public class Value_Vol : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Value_Vol

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

Value_Vol(double, string)

```
public Value_Vol(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Value_Vt

Namespace: [Csra.Settings.TheHdw.Digital.Pins.Levels](#)

Assembly: Csra.dll

```
public class Value_Vt : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Value_Vt

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

Value_Vt(double, string)

```
public Value_Vt(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Connect

Namespace: [Csra.Settings.TheHdw.Digital.Pins](#)

Assembly: Csra.dll

```
public class Connect : Setting_bool, ISetting
```

Inheritance

[object](#) ← [SettingBase<bool>](#) ← [Setting_bool](#) ← Connect

Implements

[ISetting](#)

Inherited Members

[SettingBase<bool>.pins](#) , [SettingBase<bool>.unit](#) ,
[SettingBase<bool>.SetArguments\(bool, string, bool\)](#) , [SettingBase<bool>.SetArguments\(bool, string\)](#) ,
[SettingBase<bool>.SetBehavior\(bool, string, InitMode, bool\)](#) ,
[SettingBase<bool>.SetContext\(Action<string, bool>, Func<string, bool\[\]>, Dictionary<string, bool>\)](#) ,
[SettingBase<bool>.CompareValue\(bool, bool\)](#) , [SettingBase<bool>.Validate\(\)](#) ,
[SettingBase<bool>.SerializeValue\(bool\)](#) ,
[SettingBase<bool>.SetCacheInternal\(bool, string, Dictionary<string, bool>\)](#) ,
[SettingBase<bool>.Apply\(\)](#) , [SettingBase<bool>.Diff\(\)](#) , [SettingBase<bool>.ToString\(\)](#) ,
[SettingBase<bool>.Init\(InitMode\)](#) , [SettingBase<bool>.Dump\(\)](#) , [SettingBase<bool>.Export\(string\)](#) ,
[SettingBase<bool>.Indent\(int\)](#) , [SettingBase<bool>.ParseEnum<TEnum>\(string\)](#) ,
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) ,
[object.GetHashCode\(\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Connect(bool, string)

```
public Connect(bool value, string pinList)
```

Parameters

value [bool](#)

pinList [string](#)

Methods

SetCache(bool, string)

```
public static void SetCache(bool value, string pinList)
```

Parameters

value [bool](#)

pinList [string](#)

Class InitState

Namespace: [Csra.Settings.TheHdw.Digital.Pins](#)

Assembly: Csra.dll

```
public class InitState : Setting_Enum<ChInitState>, ISetting
```

Inheritance

[object](#) ← [SettingBase](#)<ChInitState> ← [Setting_Enum](#)<ChInitState> ← InitState

Implements

[ISetting](#)

Inherited Members

[Setting_Enum<ChInitState>.SerializeValue\(ChInitState\)](#), [SettingBase<ChInitState>.pins](#),
[SettingBase<ChInitState>.unit](#), [SettingBase<ChInitState>.SetArguments\(ChInitState, string, bool\)](#),
[SettingBase<ChInitState>.SetArguments\(ChInitState, string\)](#),
[SettingBase<ChInitState>.SetBehavior\(ChInitState, string, InitMode, bool\)](#),
[SettingBase<ChInitState>.SetContext\(Action<string, ChInitState>, Func<string, ChInitState\[\]>, Dictionary<string, ChInitState>\)](#),
[SettingBase<ChInitState>.CompareValue\(ChInitState, ChInitState\)](#), [SettingBase<ChInitState>.Validate\(\)](#),
[SettingBase<ChInitState>.SetCacheInternal\(ChInitState, string, Dictionary<string, ChInitState>\)](#),
[SettingBase<ChInitState>.Apply\(\)](#), [SettingBase<ChInitState>.Diff\(\)](#), [SettingBase<ChInitState>.ToString\(\)](#),
[SettingBase<ChInitState>.Init\(InitMode\)](#), [SettingBase<ChInitState>.Dump\(\)](#),
[SettingBase<ChInitState>.Export\(string\)](#), [SettingBase<ChInitState>.Indent\(int\)](#),
[SettingBase<ChInitState>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

InitState(ChInitState, string)

```
public InitState(ChInitState value, string pinList)
```

Parameters

value ChInitState

pinList string ↗

Methods

SetCache(ChInitState, string)

```
public static void SetCache(ChInitState value, string pinList)
```

Parameters

value ChInitState

pinList string ↗

Class StartState

Namespace: [Csra.Settings.TheHdw.Digital.Pins](#)

Assembly: Csra.dll

```
public class StartState : Setting_Enum<ChStartState>, ISetting
```

Inheritance

[object](#) ← [SettingBase](#)<ChStartState> ← [Setting_Enum](#)<ChStartState> ← StartState

Implements

[ISetting](#)

Inherited Members

[Setting_Enum<ChStartState>.SerializeValue\(ChStartState\)](#), [SettingBase<ChStartState>.pins](#),
[SettingBase<ChStartState>.unit](#), [SettingBase<ChStartState>.SetArguments\(ChStartState, string, bool\)](#),
[SettingBase<ChStartState>.SetArguments\(ChStartState, string\)](#),
[SettingBase<ChStartState>.SetBehavior\(ChStartState, string, InitMode, bool\)](#),
[SettingBase<ChStartState>.SetContext\(Action<string, ChStartState>, Func<string, ChStartState\[\]>, Dictionary<string, ChStartState>\)](#),
[SettingBase<ChStartState>.CompareValue\(ChStartState, ChStartState\)](#),
[SettingBase<ChStartState>.Validate\(\)](#),
[SettingBase<ChStartState>.SetCacheInternal\(ChStartState, string, Dictionary<string, ChStartState>\)](#),
[SettingBase<ChStartState>.Apply\(\)](#), [SettingBase<ChStartState>.Diff\(\)](#),
[SettingBase<ChStartState>.ToString\(\)](#), [SettingBase<ChStartState>.Init\(InitMode\)](#),
[SettingBase<ChStartState>.Dump\(\)](#), [SettingBase<ChStartState>.Export\(string\)](#),
[SettingBase<ChStartState>.Indent\(int\)](#), [SettingBase<ChStartState>.ParseEnum<TEnum>\(string\)](#),
[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#),
[object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

StartState(ChStartState, string)

```
public StartState(ChStartState value, string pinList)
```

Parameters

value ChStartState

pinList string ↗

Methods

SetCache(ChStartState, string)

```
public static void SetCache(ChStartState value, string pinList)
```

Parameters

value ChStartState

pinList string ↗

Namespace Csra.Settings.TheHdw.Ppmu.Pins

Classes

[ClampVHi](#)

[ClampVLo](#)

[Connect](#)

[ForceV](#)

[Gate](#)

Class ClampVHi

Namespace: [Csra.Settings.TheHdw.Ppmu.Pins](#)

Assembly: Csra.dll

```
public class ClampVHi : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← ClampVHi

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

ClampVHi(double, string)

```
public ClampVHi(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class ClampVLo

Namespace: [Csra.Settings.TheHdw.Ppmu.Pins](#)

Assembly: Csra.dll

```
public class ClampVLo : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← ClampVLo

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

ClampVLo(double, string)

```
public ClampVLo(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Connect

Namespace: [Csra.Settings.TheHdw.Ppmu.Pins](#)

Assembly: Csra.dll

```
public class Connect : Setting_bool, ISetting
```

Inheritance

[object](#) ← [SettingBase<bool>](#) ← [Setting_bool](#) ← Connect

Implements

[ISetting](#)

Inherited Members

[SettingBase<bool>.pins](#) , [SettingBase<bool>.unit](#) ,
[SettingBase<bool>.SetArguments\(bool, string, bool\)](#) , [SettingBase<bool>.SetArguments\(bool, string\)](#) ,
[SettingBase<bool>.SetBehavior\(bool, string, InitMode, bool\)](#) ,
[SettingBase<bool>.SetContext\(Action<string, bool>, Func<string, bool\[\]>, Dictionary<string, bool>\)](#) ,
[SettingBase<bool>.CompareValue\(bool, bool\)](#) , [SettingBase<bool>.Validate\(\)](#) ,
[SettingBase<bool>.SerializeValue\(bool\)](#) ,
[SettingBase<bool>.SetCacheInternal\(bool, string, Dictionary<string, bool>\)](#) ,
[SettingBase<bool>.Apply\(\)](#) , [SettingBase<bool>.Diff\(\)](#) , [SettingBase<bool>.ToString\(\)](#) ,
[SettingBase<bool>.Init\(InitMode\)](#) , [SettingBase<bool>.Dump\(\)](#) , [SettingBase<bool>.Export\(string\)](#) ,
[SettingBase<bool>.Indent\(int\)](#) , [SettingBase<bool>.ParseEnum<TEnum>\(string\)](#) ,
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) ,
[object.GetHashCode\(\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Connect(bool, string)

```
public Connect(bool value, string pinList)
```

Parameters

value [bool](#)

pinList [string](#)

Methods

SetCache(bool, string)

```
public static void SetCache(bool value, string pinList)
```

Parameters

value [bool](#)

pinList [string](#)

Class ForceV

Namespace: [Csra.Settings.TheHdw.Ppmu.Pins](#)

Assembly: Csra.dll

```
public class ForceV : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← ForceV

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#) , [SettingBase<double>.pins](#) , [SettingBase<double>.unit](#) ,
[SettingBase<double>.SetArguments\(double, string, bool\)](#) ,
[SettingBase<double>.SetArguments\(double, string\)](#) ,
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#) ,
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#) ,
[SettingBase<double>.CompareValue\(double, double\)](#) , [SettingBase<double>.Validate\(\)](#) ,
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#) ,
[SettingBase<double>.Apply\(\)](#) , [SettingBase<double>.Diff\(\)](#) , [SettingBase<double>.ToString\(\)](#) ,
[SettingBase<double>.Init\(InitMode\)](#) , [SettingBase<double>.Dump\(\)](#) ,
[SettingBase<double>.Export\(string\)](#) , [SettingBase<double>.Indent\(int\)](#) ,
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

ForceV(double, string)

```
public ForceV(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Methods

SetCache(double, string)

```
public static void SetCache(double value, string pinList)
```

Parameters

value [double](#)

pinList [string](#)

Class Gate

Namespace: [Csra.Settings.TheHdw.Ppmu.Pins](#)

Assembly: Csra.dll

```
public class Gate : Setting_Enum<tlOnOff>, ISetting
```

Inheritance

[object](#) ← [SettingBase](#)<tlOnOff> ← [Setting_Enum](#)<tlOnOff> ← Gate

Implements

[ISetting](#)

Inherited Members

[Setting_Enum](#)<tlOnOff>.SerializeValue(tlOnOff) , [SettingBase](#)<tlOnOff>.pins ,
[SettingBase](#)<tlOnOff>.unit , [SettingBase](#)<tlOnOff>.SetArguments(tlOnOff, string, bool) ,
[SettingBase](#)<tlOnOff>.SetArguments(tlOnOff, string) ,
[SettingBase](#)<tlOnOff>.SetBehavior(tlOnOff, string, InitMode, bool) ,
[SettingBase](#)<tlOnOff>.SetContext(Action<string, tlOnOff>, Func<string, tlOnOff[]>, Dictionary<string, tlOnOff>) ,
[SettingBase](#)<tlOnOff>.CompareValue(tlOnOff, tlOnOff) , [SettingBase](#)<tlOnOff>.Validate() ,
[SettingBase](#)<tlOnOff>.SetCacheInternal(tlOnOff, string, Dictionary<string, tlOnOff>) ,
[SettingBase](#)<tlOnOff>.Apply() , [SettingBase](#)<tlOnOff>.Diff() , [SettingBase](#)<tlOnOff>.ToString() ,
[SettingBase](#)<tlOnOff>.Init(InitMode) , [SettingBase](#)<tlOnOff>.Dump() ,
[SettingBase](#)<tlOnOff>.Export(string) , [SettingBase](#)<tlOnOff>.Indent(int) ,
[SettingBase](#)<tlOnOff>.ParseEnum<TEnum>(string) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Gate(tlOnOff, string)

```
public Gate(tlOnOff value, string pinList)
```

Parameters

value tlOnOff

pinList string ↗

Methods

SetCache(tlOnOff, string)

```
public static void SetCache(tlOnOff value, string pinList)
```

Parameters

value tlOnOff

pinList string ↗

Namespace Csra.Settings.TheHdw.Utility.Pins

Classes

[State](#)

Class State

Namespace: [Csra.Settings.TheHdw.Utility.Pins](#)

Assembly: Csra.dll

```
public class State : Setting_Enum<tlUtilBitState>, ISetting
```

Inheritance

[object](#) ← [SettingBase](#)<tlUtilBitState> ← [Setting_Enum](#)<tlUtilBitState> ← State

Implements

[ISetting](#)

Inherited Members

[Setting_Enum<tlUtilBitState>.SerializeValue\(tlUtilBitState\)](#), [SettingBase<tlUtilBitState>.pins](#),
[SettingBase<tlUtilBitState>.unit](#), [SettingBase<tlUtilBitState>.SetArguments\(tlUtilBitState, string, bool\)](#),
[SettingBase<tlUtilBitState>.SetArguments\(tlUtilBitState, string\)](#),
[SettingBase<tlUtilBitState>.SetBehavior\(tlUtilBitState, string, InitMode, bool\)](#),
[SettingBase<tlUtilBitState>.SetContext\(Action<string, tlUtilBitState>, Func<string, tlUtilBitState\[\]>, Dictionary<string, tlUtilBitState>\)](#),
[SettingBase<tlUtilBitState>.CompareValue\(tlUtilBitState, tlUtilBitState\)](#),
[SettingBase<tlUtilBitState>.Validate\(\)](#),
[SettingBase<tlUtilBitState>.SetCacheInternal\(tlUtilBitState, string, Dictionary<string, tlUtilBitState>\)](#),
[SettingBase<tlUtilBitState>.Apply\(\)](#), [SettingBase<tlUtilBitState>.Diff\(\)](#),
[SettingBase<tlUtilBitState>.ToString\(\)](#), [SettingBase<tlUtilBitState>.Init\(InitMode\)](#),
[SettingBase<tlUtilBitState>.Dump\(\)](#), [SettingBase<tlUtilBitState>.Export\(string\)](#),
[SettingBase<tlUtilBitState>.Indent\(int\)](#), [SettingBase<tlUtilBitState>.ParseEnum<TEnum>\(string\)](#),
[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#),
[object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

State(tlUtilBitState, string)

```
public State(tlUtilBitState value, string pinList)
```

Parameters

value tlUtilBitState

pinList string ↗

Methods

SetCache(tlUtilBitState, string)

```
public static void SetCache(tlUtilBitState value, string pinList)
```

Parameters

value tlUtilBitState

pinList string ↗

Class SetSettlingTimer

Namespace: [Csra.Settings.TheHdw](#)

Assembly: Csra.dll

```
public class SetSettlingTimer : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← SetSettlingTimer

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#), [SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Constructors

SetSettlingTimer(double)

```
public SetSettlingTimer(double value)
```

Parameters

value [double](#)

Class SettleWait

Namespace: [Csra.Settings.TheHdw](#)

Assembly: Csra.dll

```
public class SettleWait : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← SettleWait

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#) , [SettingBase<double>.pins](#) , [SettingBase<double>.unit](#) ,
[SettingBase<double>.SetArguments\(double, string, bool\)](#) ,
[SettingBase<double>.SetArguments\(double, string\)](#) ,
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#) ,
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#) ,
[SettingBase<double>.CompareValue\(double, double\)](#) , [SettingBase<double>.Validate\(\)](#) ,
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#) ,
[SettingBase<double>.Apply\(\)](#) , [SettingBase<double>.Diff\(\)](#) , [SettingBase<double>.ToString\(\)](#) ,
[SettingBase<double>.Init\(InitMode\)](#) , [SettingBase<double>.Dump\(\)](#) ,
[SettingBase<double>.Export\(string\)](#) , [SettingBase<double>.Indent\(int\)](#) ,
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

SettleWait(double)

```
public SettleWait(double value)
```

Parameters

value double ↗

Class Wait

Namespace: [Csra.Settings.TheHdw](#)

Assembly: Csra.dll

```
public class Wait : Setting_double, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← [Setting_double](#) ← Wait

Implements

[ISetting](#)

Inherited Members

[Setting_double.SerializeValue\(double\)](#) , [SettingBase<double>.pins](#) , [SettingBase<double>.unit](#) ,
[SettingBase<double>.SetArguments\(double, string, bool\)](#) ,
[SettingBase<double>.SetArguments\(double, string\)](#) ,
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#) ,
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#) ,
[SettingBase<double>.CompareValue\(double, double\)](#) , [SettingBase<double>.Validate\(\)](#) ,
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#) ,
[SettingBase<double>.Apply\(\)](#) , [SettingBase<double>.Diff\(\)](#) , [SettingBase<double>.ToString\(\)](#) ,
[SettingBase<double>.Init\(InitMode\)](#) , [SettingBase<double>.Dump\(\)](#) ,
[SettingBase<double>.Export\(string\)](#) , [SettingBase<double>.Indent\(int\)](#) ,
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) ,
[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Wait(double)

```
public Wait(double value)
```

Parameters

value [double](#)

Class Custom<T>

Namespace: [Csra.Settings](#)

Assembly: Csra.dll

```
public class Custom<T> : SettingBase<T>, ISetting
```

Type Parameters

T

Inheritance

[object](#) ← [SettingBase](#)<T> ← Custom<T>

Implements

[ISetting](#)

Inherited Members

[SettingBase<T>.pins](#) , [SettingBase<T>.unit](#) , [SettingBase<T>.SetArguments\(T, string, bool\)](#) ,
[SettingBase<T>.SetArguments\(T, string\)](#) , [SettingBase<T>.SetBehavior\(T, string, InitMode, bool\)](#) ,
[SettingBase<T>.SetContext\(Action<string, T>, Func<string, T\[\]>, Dictionary<string, T>\)](#) ,
[SettingBase<T>.CompareValue\(T, T\)](#) , [SettingBase<T>.Validate\(\)](#) , [SettingBase<T>.SerializeValue\(T\)](#) ,
[SettingBase<T>.SetCacheInternal\(T, string, Dictionary<string, T>\)](#) , [SettingBase<T>.Apply\(\)](#) ,
[SettingBase<T>.Diff\(\)](#) , [SettingBase<T>.ToString\(\)](#) , [SettingBase<T>.Init\(InitMode\)](#) ,
[SettingBase<T>.Dump\(\)](#) , [SettingBase<T>.Export\(string\)](#) , [SettingBase<T>.Indent\(int\)](#) ,
[SettingBase<T>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Constructors

Custom(string, T, Action<T>, Func<T[]>, T, InitMode)

```
public Custom(string key, T value, Action<T> setAction, Func<T[]> readFunc, T initialValue,  
InitMode initMode)
```

Parameters

key [string](#)

value T

setAction [Action](#)<T>

readFunc [Func](#)<T[]>

initValue T

initMode [InitMode](#)

Custom(string, T, Action<T>, T, InitMode)

public Custom(string key, T value, Action<T> setAction, T initValue, InitMode initMode)

Parameters

key [string](#)

value T

setAction [Action](#)<T>

initValue T

initMode [InitMode](#)

Custom(T, Action<T>)

public Custom(T value, Action<T> setAction)

Parameters

value T

setAction [Action](#)<T>

Methods

SetCache(T, string)

```
public static void SetCache(T value, string key)
```

Parameters

value T

key [string](#)

Class SettingBase<T>

Namespace: [Csra.Settings](#)

Assembly: Csra.dll

Base class for all Services.Setup settings.

```
public abstract class SettingBase<T> : ISetting
```

Type Parameters

T

The setting's type.

Inheritance

[object](#) ← SettingBase<T>

Implements

[ISetting](#)

Derived

[Custom<T>](#), [Setting_Enum<T>](#), [Setting_FlaggedEnum<T>](#), [Setting_bool](#), [Setting_double](#), [Setting_int](#)

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) ,
[object.GetHashCode\(\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Fields

_pins

```
protected List<string> _pins
```

Field Value

[List](#)<[string](#)>

_unit

```
protected string _unit
```

Field Value

[string](#)

Methods

Apply()

Applies the setting to the hardware, in case and only for those pins that need it.

```
public void Apply()
```

CompareValue(T, T)

Compares two values of the setting's type. Override if special treatment is needed.

```
protected virtual bool CompareValue(T a, T b)
```

Parameters

a T

The first value for the comparison.

b T

The second value for the comparision.

Returns

[bool](#)

Returns 'true' if equal; otherwise false.

Diff()

Reads the hardware state and compares it to the setting/cached value. Target and actual status are output.

```
public void Diff()
```

Dump()

Dumps the setting to the log output target.

```
public void Dump()
```

Export(string)

Exports the setting to a file at the specified path. The file format is implementation specific, but should be human readable.

```
public void Export(string path)
```

Parameters

path [string](#)

The file system path where the setups will be exported.

Indent(int)

```
public string Indent(int level)
```

Parameters

level [int](#)

Returns

[string ↗](#)

Init(InitMode)

Performs the specified initialization to the setting.

```
public void Init(InitMode initMode)
```

Parameters

[initMode](#) [InitMode](#)

The init mode.

ParseEnum<TEnum>(string)

```
protected static TEnum ParseEnum<TEnum>(string value) where TEnum : struct
```

Parameters

[value](#) [string ↗](#)

Returns

TEnum

Type Parameters

[TEnum](#)

SerializeValue(T)

Serializes the setting's value to a string. Override if special treatment is needed.

```
protected virtual string SerializeValue(T value)
```

Parameters

value T

The value to serialize.

Returns

[string](#) ↗

The string representation of value.

SetArguments(T, string)

Hands the setting's argument parameters to the base class. Typically used for custom settings.

```
protected void SetArguments(T value, string key)
```

Parameters

value T

The value to call setAction.

key [string](#) ↗

A key to feed the cache.

SetArguments(T, string, bool)

Hands the setting's argument parameters to the base class. Typically used for instrument-related settings with or without pin relation.

```
protected void SetArguments(T value, string pinList, bool hasPins)
```

Parameters

value T

The target value of this setting.

`pinList` [string](#)

`hasPins` [bool](#)

SetBehavior(T, string, InitMode, bool)

Hands the setting's behavior parameters to the base class.

```
protected void SetBehavior(T initialValue, string unit, InitMode initMode,  
    bool doubleReadCompare)
```

Parameters

`initialValue` T

The value this setting is being initialized to by the system.

`unit` [string](#)

The unit string for the setting's value.

`initMode` [InitMode](#)

The event that initialization of this setting in the system.

`doubleReadCompare` [bool](#)

SetCacheInternal(T, string, Dictionary<string, T>)

Updates the value for specified pins in the cache. Used to make SetupService aware of manual hardware changes.

```
protected static void SetCacheInternal(T value, string pinList, Dictionary<string, T> cache)
```

Parameters

`value` T

The new value (typically what the hardware has manually be set to).

`pinList` [string](#)

`cache` [Dictionary](#)<[string](#), T>

A reference to that static cache object per setting.

SetContext(Action<string, T>, Func<string, T[]>, Dictionary<string, T>)

Hands the setting's context parameters to the base class.

```
protected void SetContext(Action<string, T> setAction, Func<string, T[]> readFunc,  
Dictionary<string, T> staticCache)
```

Parameters

`setAction` [Action](#)<[string](#), T>

The action required to apply the setting to the system.

`readFunc` [Func](#)<[string](#), T[]>

The delegate to call to read back the setting's state from the system.

`staticCache` [Dictionary](#)<[string](#), T>

The handle to the static cache object for the setting's type.

ToString()

Gets a string representation of the setting in a human-readable form.

```
public override string ToString()
```

Returns

[string](#)

The string representation.

Validate()

Validates the setting's value. Override needed per setting.

```
public virtual void Validate()
```

Class Setting_Enum<T>

Namespace: [Csra.Settings](#)

Assembly: Csra.dll

```
public abstract class Setting_Enum<T> : SettingBase<T>, ISetting where T : Enum
```

Type Parameters

T

Inheritance

[object](#) ↗ ← [SettingBase](#)<T> ← Setting_Enum<T>

Implements

[ISetting](#)

Derived

[Mode](#), [Connect](#), [Behavior](#), [Gate](#), [Mode](#), [BleederResistor](#), [Connect](#), [Mode](#), [InitState](#), [DriverMode](#), [StartState](#), [Gate](#), [State](#)

Inherited Members

[SettingBase](#)<T>.pins, [SettingBase](#)<T>.unit, [SettingBase](#)<T>.SetArguments(T, string, bool),
[SettingBase](#)<T>.SetArguments(T, string), [SettingBase](#)<T>.SetBehavior(T, string, InitMode, bool),
[SettingBase](#)<T>.SetContext(Action<string, T>, Func<string, T[]>, Dictionary<string, T>),
[SettingBase](#)<T>.CompareValue(T, T), [SettingBase](#)<T>.Validate(),
[SettingBase](#)<T>.SetCacheInternal(T, string, Dictionary<string, T>), [SettingBase](#)<T>.Apply(),
[SettingBase](#)<T>.Diff(), [SettingBase](#)<T>.ToString(), [SettingBase](#)<T>.Init(InitMode),
[SettingBase](#)<T>.Dump(), [SettingBase](#)<T>.Export(string), [SettingBase](#)<T>.Indent(int),
[SettingBase](#)<T>.ParseEnum<TEnum>(string), [object.Equals\(object\)](#) ↗, [object.Equals\(object, object\)](#) ↗,
[object.ReferenceEquals\(object, object\)](#) ↗, [object.GetHashCode\(\)](#) ↗, [object.GetType\(\)](#) ↗,
[object.MemberwiseClone\(\)](#) ↗

Methods

SerializeValue(T)

Serializes the setting's value to a string. Override if special treatment is needed.

```
protected override string SerializeValue(T value)
```

Parameters

value T

The value to serialize.

Returns

[string](#) ↗

The string representation of value.

Class Setting_FlaggedEnum<T>

Namespace: [Csra.Settings](#)

Assembly: Csra.dll

```
public abstract class Setting_FlaggedEnum<T> : SettingBase<T>, ISetting where T : Enum
```

Type Parameters

T

Inheritance

[object](#) ↗ ← [SettingBase](#)<T> ← [Setting_FlaggedEnum](#)<T>

Implements

[ISetting](#)

Inherited Members

[SettingBase](#)<T>.pins , [SettingBase](#)<T>.unit , [SettingBase](#)<T>.SetArguments(T, string, bool) ,
[SettingBase](#)<T>.SetArguments(T, string) , [SettingBase](#)<T>.SetBehavior(T, string, InitMode, bool) ,
[SettingBase](#)<T>.SetContext(Action<string, T>, Func<string, T[]>, Dictionary<string, T>) ,
[SettingBase](#)<T>.CompareValue(T, T) , [SettingBase](#)<T>.Validate() ,
[SettingBase](#)<T>.SetCacheInternal(T, string, Dictionary<string, T>) , [SettingBase](#)<T>.Apply() ,
[SettingBase](#)<T>.Diff() , [SettingBase](#)<T>.ToString() , [SettingBase](#)<T>.Init(InitMode) ,
[SettingBase](#)<T>.Dump() , [SettingBase](#)<T>.Export(string) , [SettingBase](#)<T>.Indent(int) ,
[SettingBase](#)<T>.ParseEnum<TEnum>(string) , [object.Equals\(object\)](#)↗ , [object.Equals\(object, object\)](#)↗ ,
[object.ReferenceEquals\(object, object\)](#)↗ , [object.GetHashCode\(\)](#)↗ , [object.GetType\(\)](#)↗ ,
[object.MemberwiseClone\(\)](#)↗

Methods

SerializeValue(T)

Serializes the setting's value to a string. Override if special treatment is needed.

```
protected override string SerializeValue(T value)
```

Parameters

value T

The value to serialize.

Returns

[string](#) ↗

The string representation of value.

Class Setting_bool

Namespace: [Csra.Settings](#)

Assembly: Csra.dll

```
public abstract class Setting_bool : SettingBase<bool>, ISetting
```

Inheritance

[object](#) ← [SettingBase<bool>](#) ← Setting_bool

Implements

[ISetting](#)

Derived

[Gate](#), [Connect](#), [Connect](#)

Inherited Members

[SettingBase<bool>.pins](#), [SettingBase<bool>.unit](#),
[SettingBase<bool>.SetArguments\(bool, string, bool\)](#), [SettingBase<bool>.SetArguments\(bool, string\)](#),
[SettingBase<bool>.SetBehavior\(bool, string, InitMode, bool\)](#),
[SettingBase<bool>.SetContext\(Action<string, bool>, Func<string, bool\[\]>, Dictionary<string, bool>\)](#),
[SettingBase<bool>.CompareValue\(bool, bool\)](#), [SettingBase<bool>.Validate\(\)](#),
[SettingBase<bool>.SerializeValue\(bool\)](#),
[SettingBase<bool>.SetCacheInternal\(bool, string, Dictionary<string, bool>\)](#),
[SettingBase<bool>.Apply\(\)](#), [SettingBase<bool>.Diff\(\)](#), [SettingBase<bool>.ToString\(\)](#),
[SettingBase<bool>.Init\(InitMode\)](#), [SettingBase<bool>.Dump\(\)](#), [SettingBase<bool>.Export\(string\)](#),
[SettingBase<bool>.Indent\(int\)](#), [SettingBase<bool>.ParseEnum<TEnum>\(string\)](#),
[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#),
[object.GetHashCode\(\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Class Setting_double

Namespace: [Csra.Settings](#)

Assembly: Csra.dll

```
public abstract class Setting_double : SettingBase<double>, ISetting
```

Inheritance

[object](#) ← [SettingBase<double>](#) ← Setting_double

Implements

[ISetting](#)

Derived

[CurrentLoad](#), [ComplianceRange_Negative](#), [ComplianceRange_Positive](#), [Current](#), [CurrentRange](#), [Timeout](#), [NominalBandwidth](#), [Voltage](#), [VoltageRange](#), [Level](#), [Level](#), [Level](#), [Level](#), [CurrentRange](#), [Voltage](#), [VoltageRange](#), [Value_Ioh](#), [Value_Iol](#), [Value_Vch](#), [Value_Vcl](#), [Value_Vih](#), [Value_Vil](#), [Value_Voh](#), [Value_Vol](#), [Value_Vt](#), [ClampVHi](#), [ClampVLo](#), [ForceV](#), [SetSettlingTimer](#), [SettleWait](#), [Wait](#)

Inherited Members

[SettingBase<double>.pins](#), [SettingBase<double>.unit](#),
[SettingBase<double>.SetArguments\(double, string, bool\)](#),
[SettingBase<double>.SetArguments\(double, string\)](#),
[SettingBase<double>.SetBehavior\(double, string, InitMode, bool\)](#),
[SettingBase<double>.SetContext\(Action<string, double>, Func<string, double\[\]>, Dictionary<string, double>\)](#),
[SettingBase<double>.CompareValue\(double, double\)](#), [SettingBase<double>.Validate\(\)](#),
[SettingBase<double>.SetCacheInternal\(double, string, Dictionary<string, double>\)](#),
[SettingBase<double>.Apply\(\)](#), [SettingBase<double>.Diff\(\)](#), [SettingBase<double>.ToString\(\)](#),
[SettingBase<double>.Init\(InitMode\)](#), [SettingBase<double>.Dump\(\)](#),
[SettingBase<double>.Export\(string\)](#), [SettingBase<double>.Indent\(int\)](#),
[SettingBase<double>.ParseEnum<TEnum>\(string\)](#), [object.Equals\(object\)](#),
[object.Equals\(object, object\)](#), [object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#),
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#)

Methods

[SerializeValue\(double\)](#)

Serializes the setting's value to a string. Override if special treatment is needed.

```
protected override string SerializeValue(double value)
```

Parameters

value [double](#)

The value to serialize.

Returns

[string](#)

The string representation of value.

Class Setting_int

Namespace: [Csra.Settings](#)

Assembly: Csra.dll

```
public abstract class Setting_int : SettingBase<int>, ISetting
```

Inheritance

[object](#) ← [SettingBase<int>](#) ← Setting_int

Implements

[ISetting](#)

Inherited Members

[SettingBase<int>.pins](#) , [SettingBase<int>.unit](#) , [SettingBase<int>.SetArguments\(int, string, bool\)](#) ,
[SettingBase<int>.SetArguments\(int, string\)](#) , [SettingBase<int>.SetBehavior\(int, string, InitMode, bool\)](#) ,
[SettingBase<int>.SetContext\(Action<string, int>, Func<string, int\[\]>, Dictionary<string, int>\)](#) ,
[SettingBase<int>.CompareValue\(int, int\)](#) , [SettingBase<int>.Validate\(\)](#) ,
[SettingBase<int>.SetCacheInternal\(int, string, Dictionary<string, int>\)](#) , [SettingBase<int>.Apply\(\)](#) ,
[SettingBase<int>.Diff\(\)](#) , [SettingBase<int>.ToString\(\)](#) , [SettingBase<int>.Init\(InitMode\)](#) ,
[SettingBase<int>.Dump\(\)](#) , [SettingBase<int>.Export\(string\)](#) , [SettingBase<int>.Indent\(int\)](#) ,
[SettingBase<int>.ParseEnum<TEnum>\(string\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

SerializeValue(int)

Serializes the setting's value to a string. Override if special treatment is needed.

```
protected override string SerializeValue(int value)
```

Parameters

value [int](#)

The value to serialize.

Returns

[string](#) ↗

The string representation of value.

Namespace Csra.TheLib

Namespaces

[Csra.TheLib.Acquire](#)

[Csra.TheLib.Execute](#)

[Csra.TheLib.Setup](#)

Classes

[Datalog](#)

[Validate](#)

Namespace Csra.TheLib.Acquire

Classes

[Dc](#)

[Digital](#)

[ScanNetwork](#)

[Search](#)

Class Dc

Namespace: [Csra.TheLib.Acquire](#)

Assembly: Csra.dll

```
public class Dc : ILib.IAcquire.IDc
```

Inheritance

[object](#) ← Dc

Implements

[ILib.IAcquire.IDc](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Measure(Pins, int, double?, Measure?)

Performs multiple measurements for the set of pins provided.

```
public PinSite<double> Measure(Pins pins, int sampleSize, double? sampleRate = null,  
Measure? meterMode = null)
```

Parameters

pins [Pins](#)

The pins to measure.

sampleSize [int](#)

The number of samples.

sampleRate [double](#)?

Optional. The sampling rate.

`meterMode` [Measure?](#)

Optional. Set the mode to measure Voltage or Current.

Returns

`PinSite<double>`

Returns an average value.

Exceptions

[Exception](#)

Appears when pinList contains different types of pins - temporary limitation in functionality.

Measure(Pins, Measure?)

Performs a single measurement for the set of pins provided.

```
public PinSite<double> Measure(Pins pins, Measure? meterMode = null)
```

Parameters

`pins` [Pins](#)

The pins to measure.

`meterMode` [Measure?](#)

Optional. Set the mode to measure Voltage or Current.

Returns

`PinSite<double>`

Returns a value.

Exceptions

Exception

Appears when pinList contains different types of pins - temporary limitation in functionality.

Measure(Pins[], int[], double[], Measure[])

Performs multiple measurements for the set of each element in pinGroups.

```
public PinSite<double> Measure(Pins[] pinGroups, int[] sampleSizes, double[] sampleRates = null, Measure[] meterModes = null)
```

Parameters

pinGroups [Pins\[\]](#)

Array of pin or pin groups.

sampleSizes [int\[\]](#)

Array of number of samples.

sampleRates [double\[\]](#)

Optional. Array of sampling rate.

meterModes [Measure\[\]](#)

Optional. Array of settings measurements mode voltage and current.

Returns

[PinSite<double>](#)

Returns a set of measurements.

Exceptions

Exception

Appears when an element of pinGroups contains different types of pins - temporary limitation in functionality.

MeasureSamples(Pins, int, double?, Measure?)

Performs multiple measurements for the set of pins provided.

```
public PinSite<Samples<double>> MeasureSamples(Pins pins, int sampleSize, double? sampleRate = null, Measure? meterMode = null)
```

Parameters

pins [Pins](#)

The pins to measure.

sampleSize [int](#)

The number of samples.

sampleRate [double](#)?

Optional. The sampling rate.

meterMode [Measure](#)?

Optional. Set the mode to measure Voltage or Current.

Returns

[PinSite<Samples<double>>](#)

Returns a set of measurements.

Exceptions

[Exception](#)

Appears when pinList contains different types of pins - temporary limitation in functionality.

MeasureSamples(Pins[], int[], double[], Measure[])

Performs multiple measurements for the set of each element in pinGroups.

```
public PinSite<Samples<double>> MeasureSamples(Pins[] pinGroups, int[] sampleSizes, double[] sampleRates = null, Measure[] meterModes = null)
```

Parameters

pinGroups [Pins\[\]](#)

Array of pin or pin groups.

sampleSizes [int\[\]](#)

Array of number of samples.

sampleRates [double\[\]](#)

Optional. Array of sampling rate.

meterModes [Measure\[\]](#)

Optional. Array of settings measurements mode voltage and current.

Returns

[PinSite<Samples<double>>](#)

Returns a set of measurements.

Exceptions

[Exception](#)

Appears when an element of pinGroups contains different types of pins - temporary limitation in functionality.

ReadCaptured(Pins, string)

Allows configuration and control of capture parameters for the specified pins.

```
public PinSite<double> ReadCaptured(Pins pins, string signalName)
```

Parameters

pins [Pins](#)

List of pin or pin group names.

signalName [string](#)

The name of the read signal.

Returns

[PinSite<double>](#)

Returns a value.

Exceptions

[Exception](#)

Appears when pinList contains different types of pins - temporary limitation in functionality.

ReadCapturedSamples(Pins, string)

Allows configuration and control of capture parameters for the specified pins.

```
public PinSite<Samples<double>> ReadCapturedSamples(Pins pins, string signalName)
```

Parameters

pins [Pins](#)

List of pin or pin group names.

signalName [string](#)

The name of the read signal.

Returns

[PinSite<Samples<double>>](#)

Returns a set of measurements.

Exceptions

[Exception ↗](#)

Appears when pinList contains different types of pins - temporary limitation in functionality.

ReadMeasured(Pins, int, double?)

Performs multiple readings of measurements, depending on the sample size parameter, for the set of pins provided.

```
public PinSite<double> ReadMeasured(Pins pins, int sampleSize, double? sampleRate = null)
```

Parameters

[pins Pins](#)

The pins to read measurements.

[sampleSize int ↗](#)

The number of samples.

[sampleRate double ↗?](#)

Optional. The sampling rate.

Returns

[PinSite<double ↗>](#)

Returns an average value.

Exceptions

[Exception ↗](#)

Appears when pinList contains different types of pins - temporary limitation in functionality.

ReadMeasuredSamples(Pins, int, double?)

Performs multiple readings of measurements, depending on the sample size parameter, for the set of pins provided.

```
public PinSite<Samples<double>> ReadMeasuredSamples(Pins pins, int sampleSize, double?  
sampleRate = null)
```

Parameters

pins [Pins](#)

The pins to read measurements.

sampleSize [int](#)

The number of samples.

sampleRate [double](#)?

Optional. The sampling rate.

Returns

[PinSite<Samples<double>>](#)

Returns a set of measurements.

Exceptions

[Exception](#)

Appears when pinList contains different types of pins - temporary limitation in functionality.

Strobe(Pins)

Performs a single measurement (strobe) on the according instrument, to read back the value later.

```
public void Strobe(Pins pins)
```

Parameters

pins [Pins](#)

The pins to be measured.

StrobeSamples(Pins, int, double?)

Performs multiple measurements (strokes) on the according instrument, to read back the value later.

```
public void StrobeSamples(Pins pins, int sampleSize, double? sampleRate = null)
```

Parameters

pins [Pins](#)

The pins to be measured.

sampleSize [int](#)

Number of measurements (Strobes).Ignored for PPMU.

sampleRate [double](#)?

Optional. The sampling rate. Default is the current hardware setting. Ignored for PPMU.

Class Digital

Namespace: [Csra.TheLib.Acquire](#)

Assembly: Csra.dll

```
public class Digital : ILib.IAcquire.IDigital
```

Inheritance

[object](#) ← Digital

Implements

[ILib.IAcquire.IDigital](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

MeasureFrequency(Pins)

Measures and returns the frequency configured by Setup.Digital.FrequencyCounter.

```
public PinSite<double> MeasureFrequency(Pins pins)
```

Parameters

pins [Pins](#)

List of pin or pin group names.

Returns

[PinSite<double>](#)

The measured frequency for each pin in Hz.

PatternResults()

```
public Site<bool> PatternResults()
```

Returns

Site<[bool](#)>

Read(Pins, int, int)

Reads captured pin data from HRAM and returns raw results as IPinListData

```
public PinSite<Samples<int>> Read(Pins pins, int startIndex = 0, int cycle = 0)
```

Parameters

[pins](#) [Pins](#)

Pin names, must contain digital pins

[startIndex](#) [int](#)

Optional. Index to start capture

[cycle](#) [int](#)

Optional. Cycle of data to capture for pins in 2x or 4x modes

Returns

PinSite<Samples<[int](#)>>

Raw captured pin data

Exceptions

[ArgumentException](#)

ReadWords(Pins, int, int, int, tlBitOrder)

Reads pin data from specified cycles in HRAM for the specified pin, groups the data into words of a specified size, and populates these words into an array of SiteLong objects

```
public PinSite<Samples<int>> ReadWords(Pins pins, int startIndex, int length, int wordSize, tlBitOrder bitOrder)
```

Parameters

pins [Pins](#)

Pin names, digital pins required

startIndex [int](#)

Index to start data processing

length [int](#)

Number of bits to process

wordSize [int](#)

Number of bits in each word

bitOrder [tlBitOrder](#)

Order of bits, either msbFirst or lsbFirst

Returns

[PinSite<Samples<int>>](#)

Word values in array of ISiteLong

Exceptions

[ArgumentException](#)

Class ScanNetwork

Namespace: [Csra.TheLib.Acquire](#)

Assembly: Csra.dll

```
public class ScanNetwork : ILib.IAcquire.IScanNetwork
```

Inheritance

[object](#) ← ScanNetwork

Implements

[ILib.IAcquire.IScanNetwork](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

PatternResults(ScanNetworkPatternInfo)

Retrieve the per core/icl instance results of the specified ScanNetwork pattern object from its latest execution (TheLib.Execute.ScanNetwork.RunPattern(ScanNetworkPatternObject)).

```
public ScanNetworkPatternResults PatternResults(ScanNetworkPatternInfo scanNetworkPattern)
```

Parameters

scanNetworkPattern [ScanNetworkPatternInfo](#)

The [ScanNetworkPatternInfo](#) Object that is associated with the ScanNetwork pattern(set).

Returns

[ScanNetworkPatternResults](#)

[ScanNetworkPatternResults](#) object that contains per core/icl instance results

Class Search

Namespace: [Csra.TheLib.Acquire](#)

Assembly: Csra.dll

```
public class Search : ILib.IAcquire.ISearch
```

Inheritance

[object](#) ← Search

Implements

[ILib.IAcquire.ISearch](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

**BinarySearch<Tout>(double, double, double, bool,
Func<Site<double>, Site<Tout>>, Func<Tout, bool>, double)**

Performs a floating point binary search between `inFrom` and `inTo` by executing `oneMeasurement` at each step. Determines the input resulting in an output closest to the trip criteria's inflection point. The number of steps executed equals $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the minimum delta is reached, the reported result lies within `inMinDelta` from the ideal result, matching criteria's direction. If no transition was found, the method returns `inNotFoundResult`.

```
public Site<double> BinarySearch<Tout>(double inFrom, double inTo, double inMinDelta, bool  
invertingLogic, Func<Site<double>, Site<Tout>> oneMeasurement, Func<Tout, bool>  
outTripCriteria, double inNotFoundResult)
```

Parameters

`inFrom` [double](#)

The lower boundary of the search range. Must be less than `inTo`.

`inTo` [double](#)

The upper boundary of the search range. Must be greater than `inFrom`

`inMinDelta` [double](#)

The minimum allowable difference between successive input values, used to determine when the search should stop.

`invertingLogic` [bool](#)

Determines whether the logic is inverted. `false` means higher input values increase the likelihood of meeting the trip criteria. `true` means lower input values do.

`oneMeasurement` [Func](#)<`Site<double>`, `Site<Tout>`>

The action to execute for every measurement.

`outTripCriteria` [Func](#)<`Tout`, [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

`inNotFoundResult` [double](#)

The return value for the case when the trip criteria was never found.

Returns

`Site<double>`

The input value resulting in an output fulfilling the trip criteria and be closest to it's inflection point.
The worst case deviation from the exact input is (`inMinDelta`.

Type Parameters

`Tout`

The type of the device's output.

`BinarySearch<Tout>(double, double, double, bool,
Func<Site<double>, Site<Tout>>, Func<Tout, bool>, double,
out Site<Tout>)`

Performs a floating point binary search between `inFrom` and `inTo` by executing `oneMeasurement` at each step. Determines the input resulting in an output closest to the trip criteria's inflection point. The number of steps executed equals $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the minimum delta is reached, the reported result lies within `inMinDelta` from the ideal result, matching criteria's direction. Additionally provides the output value of the input step found. If no transition was found, the method returns `inNotFoundResult`.

```
public Site<double> BinarySearch<Tout>(double inFrom, double inTo, double inMinDelta, bool invertingLogic, Func<Site<double>, Site<Tout>> oneMeasurement, Func<Tout, bool> outTripCriteria, double inNotFoundResult, out Site<Tout> outResult)
```

Parameters

`inFrom` [double](#)

The lower boundary of the search range. Must be less than `inTo`.

`inTo` [double](#)

The upper boundary of the search range. Must be greater than `inFrom`

`inMinDelta` [double](#)

The minimum allowable difference between successive input values, used to determine when the search should stop.

`invertingLogic` [bool](#)

Determines whether the logic is inverted. `false` means higher input values increase the likelihood of meeting the trip criteria. `true` means lower input values do.

`oneMeasurement` [Func](#)<Site<[double](#)>, Site<Tout>>

The action to execute for every measurement.

`outTripCriteria` [Func](#)<Tout, [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

`inNotFoundResult` [double](#)

The return value for the case when the trip criteria was never found.

`outResult` Site<Tout>

Output - contains the output value of the input found.

Returns

Site<[double](#)>

The input value resulting in an output fulfilling the trip criteria and be closest to it's inflection point.

The worst case deviation from the exact input is ([inMinDelta](#)).

Type Parameters

Tout

The type of the device's output.

BinarySearch<Tout>(double, double, double, bool,
Func<Site<double>, Site<Tout>>, Tout)

Performs a floating point binary search between [inFrom](#) and [inTo](#) by executing [oneMeasurement](#) at each step. Determines the input resulting in an output closest to the (numeric) target. The number of steps executed equals $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the minimum delta is reached, the reported result lies within +/- [inMinDelta](#) from the ideal result.

```
public Site<double> BinarySearch<Tout>(double inFrom, double inTo, double inMinDelta, bool  
invertingLogic, Func<Site<double>, Site<Tout>> oneMeasurement, Tout outTarget)
```

Parameters

[inFrom](#) [double](#)

The lower boundary of the search range. Must be less than [inTo](#).

[inTo](#) [double](#)

The upper boundary of the search range. Must be greater than [inFrom](#)

[inMinDelta](#) [double](#)

The minimum allowable difference between successive input values, used to determine when the search should stop. Must be >0.

[invertingLogic](#) [bool](#)

A flag indicating whether the output is inverted, meaning increasing output values are a result of decreasing input values.

oneMeasurement [Func](#)<Site<[double](#)>, Site<Tout>>

The action to execute for every measurement.

outTarget Tout

The (numeric) target output value for which the corresponding input condition is searched.

Returns

Site<[double](#)>

The input value resulting in an output closest to the target. The worst case delta to the exact input is $\pm (\text{inMinDelta} / 2)$ if it can be reached given the search range.

Type Parameters

Tout

The type of the device's output.

BinarySearch<Tout>(double, double, double, bool, Func<Site<double>, Site<Tout>>, Tout, out Site<Tout>)

Performs a floating point binary search between `inFrom` and `inTo` by executing `oneMeasurement` at each step. Determines the input resulting in an output closest to the (numeric) target. The number of steps executed equals $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the minimum delta is reached, the reported result lies within $\pm \text{inMinDelta}$ from the ideal result. Additionally provides the output value for the input step found.

```
public Site<double> BinarySearch<Tout>(double inFrom, double inTo, double inMinDelta, bool
invertingLogic, Func<Site<double>, Site<Tout>> oneMeasurement, Tout outTarget, out
Site<Tout> outResult)
```

Parameters

inFrom [double](#)

The lower boundary of the search range. Must be less than `inTo`.

`inTo` [double](#)

The upper boundary of the search range. Must be greater than `inFrom`

`inMinDelta` [double](#)

The minimum allowable difference between successive input values, used to determine when the search should stop.

`invertingLogic` [bool](#)

A flag indicating whether the output is inverted, meaning increasing output values are a result of decreasing input values.

`oneMeasurement` [Func](#)<Site<[double](#)>, Site<`Tout`>>

The action to execute for every measurement.

`outTarget` `Tout`

The (numeric) target output value for which the corresponding input condition is searched.

`outResult` Site<`Tout`>

Output - contains the output value for the input found.

Returns

Site<[double](#)>

The input value resulting in an output closest to the target. The worst case delta to the exact input is $\pm (\text{inMinDelta} / 2)$ if it can be reached given the search range.

Type Parameters

`Tout`

The type of the device's output.

BinarySearch<`Tout`>(int, int, int, bool, Func<Site<int>, Site<`Tout`>>, Func<`Tout`, bool>, int)

Performs an integer binary search between `inFrom` and `inTo` by executing `oneMeasurement` at each step. Determines the input resulting in an output closest to the trip criteria's inflection point. The number of steps executed does not exceed $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the input value closest to the transition point (within the specified `inMinDelta` resolution), but matching the criteria is reached. If no transition was found, the method returns `inNotFoundResult`.

```
public Site<int> BinarySearch<Tout>(int inFrom, int inTo, int inMinDelta, bool
invertingLogic, Func<Site<int>, Site<Tout>> oneMeasurement, Func<Tout, bool>
outTripCriteria, int inNotFoundResult)
```

Parameters

`inFrom` [int](#)

The lower boundary of the search range. Must be less than `inTo`.

`inTo` [int](#)

The upper boundary of the search range. Must be greater than `inFrom`.

`inMinDelta` [int](#)

The minimum allowable difference between successive input values, used to determine when the search should stop.

`invertingLogic` [bool](#)

Determines whether the logic is inverted. `false` means higher input values increase the likelihood of meeting the trip criteria. `true` means lower input values do.

`oneMeasurement` [Func](#)<`Site<int>`, `Site<Tout>`>

The action to execute for every measurement.

`outTripCriteria` [Func](#)<`Tout`, [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

`inNotFoundResult` [int](#)

The return value for the case when the trip criteria was never found.

Returns

Site<[int](#)>

The input value resulting in an output fulfilling the trip criteria and be closest to it's inflection point.
The worst case deviation from the exact input is ([inMinDelta](#)).

Type Parameters

Tout

The type of the device's output.

BinarySearch<Tout>(int, int, int, bool, Func<Site<int>, Site<Tout>>, Func<Tout, bool>, int, out Site<Tout>)

Performs an integer binary search between [inFrom](#) and [inTo](#) by executing [oneMeasurement](#) at each step. Determines the input resulting in an output closest to the trip criteria's inflection point. The number of steps executed does not exceed $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the input value closest to the transition point (within the specified [inMinDelta](#) resolution), but matching the criteria is reached. If no transition was found, the method returns [inNotFoundResult](#).

```
public Site<int> BinarySearch<Tout>(int inFrom, int inTo, int inMinDelta, bool
invertingLogic, Func<Site<int>, Site<Tout>> oneMeasurement, Func<Tout, bool>
outTripCriteria, int inNotFoundResult, out Site<Tout> outResult)
```

Parameters

[inFrom](#) [int](#)

The lower boundary of the search range. Must be less than [inTo](#).

[inTo](#) [int](#)

The upper boundary of the search range. Must be greater than [inFrom](#)

[inMinDelta](#) [int](#)

The minimum allowable difference between successive input values, used to determine when the search should stop.

[invertingLogic](#) [bool](#)

Determines whether the logic is inverted. `false` means higher input values increase the likelihood of meeting the trip criteria. `true` means lower input values do.

`oneMeasurement` [Func](#)<Site<[int](#)>, Site<[Tout](#)>>

The action to execute for every measurement.

`outTripCriteria` [Func](#)<[Tout](#), [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

`inNotFoundResult` [int](#)

The return value for the case when the trip criteria was never found.

`outResult` Site<[Tout](#)>

Output - contains the output value of the input found.

Returns

Site<[int](#)>

The input value resulting in an output fulfilling the trip criteria and be closest to it's inflection point.
The worst case deviation from the exact input is (`inMinDelta`).

Type Parameters

`Tout`

The type of the device's output.

`BinarySearch<Tout>(int, int, int, bool, Func<Site<int>, Site<Tout>>, Tout)`

Performs an integer binary search between `inFrom` and `inTo` by executing `oneMeasurement` at each step. Determines the input resulting in an output closest to the target. The number of steps executed does not exceed $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the best input value (within the specified `inMinDelta` resolution) is reached.

```
public Site<int> BinarySearch<Tout>(int inFrom, int inTo, int inMinDelta, bool  
invertingLogic, Func<Site<int>, Site<Tout>> oneMeasurement, Tout outTarget)
```

Parameters

`inFrom` [int](#)

The lower boundary of the search range. Must be less than `inTo`.

`inTo` [int](#)

The upper boundary of the search range. Must be greater than `inFrom`

`inMinDelta` [int](#)

The minimum allowable difference between successive input values, used to determine when the search should stop. Must be >0.

`invertingLogic` [bool](#)

A flag indicating whether the output is inverted, meaning increasing output values are a result of decreasing input values.

`oneMeasurement` [Func](#)<`Site<int>`, `Site<Tout>`>

The action to execute for every measurement.

`outTarget` `Tout`

The (numeric) target output value for which the corresponding input condition is searched.

Returns

`Site<int>`

The input value resulting in an output closest to the target. The worst case delta to the exact input is $\pm (\text{inMinDelta} / 2)$ if it can be reached given the search range.

Type Parameters

`Tout`

The type of the device's output.

`BinarySearch<Tout>(int, int, int, bool, Func<Site<int>, Site<Tout>>, Tout, out Site<Tout>)`

Performs an integer binary search between `inFrom` and `inTo` by executing `oneMeasurement` at each step. Determines the input resulting in an output closest to the target. The number of steps executed does not exceed $\log_2((\text{inTo} - \text{inFrom}) / \text{inMinDelta})$, rounded up to the next integer. The search ends when the best input value (within the specified `inMinDelta` resolution) is reached.

```
public Site<int> BinarySearch<Tout>(int inFrom, int inTo, int inMinDelta, bool  
invertingLogic, Func<Site<int>, Site<Tout>> oneMeasurement, Tout outTarget, out  
Site<Tout> outResult)
```

Parameters

`inFrom` [int](#)

The lower boundary of the search range. Must be less than `inTo`.

`inTo` [int](#)

The upper boundary of the search range. Must be greater than `inFrom`.

`inMinDelta` [int](#)

The minimum allowable difference between successive input values, used to determine when the search should stop.

`invertingLogic` [bool](#)

A flag indicating whether the output is inverted, meaning increasing output values are a result of decreasing input values.

`oneMeasurement` [Func](#)<Site<[int](#)>, Site<Tout>>

The action to execute for every measurement.

`outTarget` [Tout](#)

The (numeric) target output value for which the corresponding input condition is searched.

`outResult` [Site](#)<Tout>

Output - contains the output value for the input found.

Returns

[Site](#)<[int](#)>

The input value resulting in an output closest to the target. The worst case delta to the exact input is $+/- (\text{inMinDelta} / 2)$ if it can be reached given the search range.

Type Parameters

Tout

The type of the device's output.

LinearFullFromIncCount<Tin>(Tin, Tin, int, Action<Tin>)

Performs a linear search from **inFrom** by increasing with **inIncrement**. Executes **oneMeasurement** at each step including the start point. The ramp ends after **inCount** measurements are completed.

```
public void LinearFullFromIncCount<Tin>(Tin inFrom, Tin inIncrement, int inCount,  
Action<Tin> oneMeasurement)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inIncrement Tin

The input increment value for every step.

inCount int

The total number of steps to execute.

oneMeasurement Action<Tin>

The action to execute for every measurement.

Type Parameters

Tin

The type of the input condition for the device.

LinearFullFromToCount<Tin>(Tin, Tin, int, Action<Tin>)

Performs a linear search between `inFrom` and `inTo` with `inCount` inputs. Executes `oneMeasurement` at each step including the end points. The ramp ends after `inCount` measurements are completed.

```
public Tin LinearFullFromToCount<Tin>(Tin inFrom, Tin inTo, int inCount,  
Action<Tin> oneMeasurement)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inTo` Tin

The end point of the linear input ramp.

`inCount` [int](#)

The number of equally spaced steps to execute, including both endpoints exactly.

`oneMeasurement` [Action](#)<Tin>

The action to execute for every measurement.

Returns

Tin

The calculated increment, for later use in the processing step (ignore if not needed).

Type Parameters

`Tin`

The type of the input condition for the device.

LinearFullFromToInc<Tin>(Tin, Tin, Tin, Action<Tin>)

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. The ramp ends with the last input less or equal `inTo`.

```
public void LinearFullFromToInc<Tin>(Tin inFrom, Tin inTo, Tin inIncrement,  
Action<Tin> oneMeasurement)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inTo Tin

The end point of the linear input ramp.

inIncrement Tin

The input increment value for every step.

oneMeasurement Action<Tin>

The action to execute for every measurement.

Type Parameters

Tin

The type of the input condition for the device.

**LinearStopFromIncCount<Tin, Tout>(Tin, Tin, int, Tin, Tin,
Func<Tin, Site<Tout>>, Func<Tout, bool>)**

Performs a linear search from **inFrom** by increasing with **inIncrement**. Executes **oneMeasurement** at each step including the start point. Determines the first input meeting the **outTripCriteria**. The ramp stops prematurely when the trip criteria is met on all sites, or after **inCount** measurements are completed.

```
public Site<Tin> LinearStopFromIncCount<Tin, Tout>(Tin inFrom, Tin inIncrement, int inCount,  
Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout,  
bool> outTripCriteria)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inIncrement `Tin`

The input increment value for every step.

inCount `int`

The total number of steps to execute.

inOffset `Tin`

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

oneMeasurement `Func<Tin, Site<Tout>>`

The action to execute for every measurement.

outTripCriteria `Func<Tout, bool>`

A delegate indicating the output meets the condition required for the input value searched.

Returns

`Site<Tin>`

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

`Tin`

The type of the input condition for the device.

`Tout`

The type of the device's output.

`LinearStopFromIncCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>, out Site<int>)`

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or after `inCount` measurements are completed. Additionally provides the index of the input step found.

```
public Site<Tin> LinearStopFromIncCount<Tin, Tout>(Tin inFrom, Tin inIncrement, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout, bool> outTripCriteria, out Site<int> tripIndex)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inIncrement` Tin

The input increment value for every step.

`inCount` [int](#)

The total number of steps to execute.

`inOffset` Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

`inNotFoundResult` Tin

The return value for the case when the trip criteria was never found.

`oneMeasurement` [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

`outTripCriteria` [Func](#)<Tout, [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

`tripIndex` Site<[int](#)>

Output - contains the index of the input step found.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromIncCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>, out Site<int>, out Site<Tout>)

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or after `inCount` measurements are completed. Additionally provides the index and output value of the input step found.

```
public Site<Tin> LinearStopFromIncCount<Tin, Tout>(Tin inFrom, Tin inIncrement, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout, bool> outTripCriteria, out Site<int> tripIndex, out Site<Tout> tripOut)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inIncrement` Tin

The input increment value for every step.

inCount [int](#)

The total number of steps to execute.

inOffset [Tin](#)

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult [Tin](#)

The return value for the case when the trip criteria was never found.

oneMeasurement [Func](#)<[Tin](#), [Site<Tout>](#)>

The action to execute for every measurement.

outTripCriteria [Func](#)<[Tout](#), [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

tripIndex [Site<int](#)>

Output - contains the index of the input step found.

tripOut [Site<Tout>](#)

Output - contains the output value of the input step found.

Returns

[Site<Tin>](#)

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

`LinearStopFromIncCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Tout)`

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or after `inCount` measurements are completed.

```
public Site<Tin> LinearStopFromIncCount<Tin, Tout>(Tin inFrom, Tin inIncrement, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inIncrement` Tin

The input increment value for every step.

`inCount` [int](#)

The total number of steps to execute.

`inOffset` Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

`inNotFoundResult` Tin

The return value for the case when the trip criteria was never found.

`oneMeasurement` [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

`outTarget` Tout

The (numeric) target output value to be searched.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromIncCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Tout, out Site<int>)

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or after `inCount` measurements are completed. Additionally provides the index of the input step found.

```
public Site<Tin> LinearStopFromIncCount<Tin, Tout>(Tin inFrom, Tin inIncrement, int inCount,
Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget,
out Site<int> closestIndex)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inIncrement Tin

The input increment value for every step.

inCount int

The total number of steps to execute.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

oneMeasurement `Func<Tin, Site<Tout>>`

The action to execute for every measurement.

outTarget `Tout`

The (numeric) target output value to be searched.

closestIndex `Site<int>`

Output - contains the index of the input step found.

Returns

`Site<Tin>`

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

`LinearStopFromIncCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Tout, out Site<int>, out Site<Tout>)`

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or after `inCount` measurements are completed. Additionally provides the index and output value of the input step found.

```
public Site<Tin> LinearStopFromIncCount<Tin, Tout>(Tin inFrom, Tin inIncrement, int inCount,
Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget,
out Site<int> closestIndex, out Site<Tout> closestOut)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inIncrement Tin

The input increment value for every step.

inCount [int](#)

The total number of steps to execute.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

oneMeasurement [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

outTarget Tout

The (numeric) target output value to be searched.

closestIndex Site<[int](#)>

Output - contains the index of the input step found.

closestOut Site<Tout>

Output - contains the output value of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>)

Performs a linear search between `inFrom` and `inTo` with `inCount` inputs. Executes `oneMeasurement` at each step including the end points. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or after `inCount` measurements are completed.

```
public Site<Tin> LinearStopFromToCount<Tin, Tout>(Tin inFrom, Tin inTo, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout, bool> outTripCriteria)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inTo Tin

The end point of the linear input ramp.

inCount int

The number of equally spaced steps to execute, including both endpoints exactly.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

oneMeasurement `Func<Tin, Site<Tout>>`

The action to execute for every measurement.

outTripCriteria `Func<Tout, bool>`

A delegate indicating the output meets the condition required for the input value searched.

Returns

`Site<Tin>`

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

`Tin`

The type of the input condition for the device.

`Tout`

The type of the device's output.

`LinearStopFromToCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>, out Site<int>)`

Performs a linear search between `inFrom` and `inTo` with `inCount` inputs. Executes `oneMeasurement` at each step including the end points. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or after `inCount` measurements are completed. Additionally provides the index of the input step found.

```
public Site<Tin> LinearStopFromToCount<Tin, Tout>(Tin inFrom, Tin inTo, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout, bool> outTripCriteria, out Site<int> tripIndex)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

inTo Tin

The end point of the linear input ramp.

inCount [int](#)

The number of equally spaced steps to execute, including both endpoints exactly.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

oneMeasurement [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

outTripCriteria [Func](#)<Tout, [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

tripIndex Site<[int](#)>

Output - contains the index of the input step found.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>, out Site<int>, out Site<Tout>)

Performs a linear search between `inFrom` and `inTo` with `inCount` inputs. Executes `oneMeasurement` at each step including the end points. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or after `inCount` measurements are completed. Additionally provides the index and output value of the input step found.

```
public Site<Tin> LinearStopFromToCount<Tin, Tout>(Tin inFrom, Tin inTo, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout, bool> outTripCriteria, out Site<int> tripIndex, out Site<Tout> tripOut)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inTo` Tin

The end point of the linear input ramp.

`inCount` [int](#)

The number of equally spaced steps to execute, including both endpoints exactly.

`inOffset` Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

`inNotFoundResult` Tin

The return value for the case when the trip criteria was never found.

`oneMeasurement` [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

outTripCriteria `Func<Tout, bool>`

A delegate indicating the output meets the condition required for the input value searched.

tripIndex `Site<int>`

Output - contains the index of the input step found.

tripOut `Site<Tout>`

Output - contains the output value of the input step found.

Returns

`Site<Tin>`

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Tout)

Performs a linear search between `inFrom` and `inTo` with `inCount` inputs. Executes `oneMeasurement` at each step including the end points. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or after `inCount` measurements are completed.

```
public Site<Tin> LinearStopFromToCount<Tin, Tout>(Tin inFrom, Tin inTo, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget)
```

Parameters

inFrom `Tin`

The starting point of the linear input ramp.

inTo `Tin`

The end point of the linear input ramp.

inCount `int`

The number of equally spaced steps to execute, including both endpoints exactly.

inOffset `Tin`

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

oneMeasurement `Func`<`Tin`, `Site`<`Tout`>>

The action to execute for every measurement.

outTarget `Tout`

The (numeric) target output value to be searched.

Returns

`Site`<`Tin`>

The input value resulting in an output closest to the target.

Type Parameters

`Tin`

The type of the input condition for the device.

`Tout`

The type of the device's output.

`LinearStopFromToCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Tout, out Site<int>)`

Performs a linear search between `inFrom` and `inTo` with `inCount` inputs. Executes `oneMeasurement` at each step including the end points. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or after `inCount` measurements are completed. Additionally provides the index of the input step found.

```
public Site<Tin> LinearStopFromToCount<Tin, Tout>(Tin inFrom, Tin inTo, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget, out Site<int> closestIndex)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inTo` Tin

The end point of the linear input ramp.

`inCount` [int](#)

The number of equally spaced steps to execute, including both endpoints exactly.

`inOffset` Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

`inNotFoundResult` Tin

The return value for the case when the trip criteria was never found.

`oneMeasurement` [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

`outTarget` Tout

The (numeric) target output value to be searched.

`closestIndex` Site<[int](#)>

Output - contains the index of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToCount<Tin, Tout>(Tin, Tin, int, Tin, Tin, Func<Tin, Site<Tout>>, Tout, out Site<int>, out Site<Tout>)

Performs a linear search between `inFrom` and `inTo` with `inCount` inputs. Executes `oneMeasurement` at each step including the end points. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or after `inCount` measurements are completed. Additionally provides the index and output value of the input step found.

```
public Site<Tin> LinearStopFromToCount<Tin, Tout>(Tin inFrom, Tin inTo, int inCount, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget, out Site<int> closestIndex, out Site<Tout> closestOut)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`inTo` Tin

The end point of the linear input ramp.

`inCount` [int](#)

The number of equally spaced steps to execute, including both endpoints exactly.

inOffset `Tin`

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

oneMeasurement `Func<Tin, Site<Tout>>`

The action to execute for every measurement.

outTarget `Tout`

The (numeric) target output value to be searched.

closestIndex `Site<int>`

Output - contains the index of the input step found.

closestOut `Site<Tout>`

Output - contains the output value of the input step found.

Returns

`Site<Tin>`

The input value resulting in an output closest to the target.

Type Parameters

`Tin`

The type of the input condition for the device.

`Tout`

The type of the device's output.

LinearStopFromToInc<Tin, Tout>(Tin, Tin, Tin, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>)

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or with the last input less or equal `inTo`.

```
public Site<Tin> LinearStopFromToInc<Tin, Tout>(Tin inFrom, Tin to, Tin inIncrement,  
Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout,  
bool> outTripCriteria)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`to` Tin

`inIncrement` Tin

The input increment value for every step.

`inOffset` Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

`inNotFoundResult` Tin

The return value for the case when the trip criteria was never found.

`oneMeasurement` [Func<Tin, Site<Tout>>](#)

The action to execute for every measurement.

`outTripCriteria` [Func<Tout, bool>](#)

A delegate indicating the output meets the condition required for the input value searched.

Returns

`Site<Tin>`

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToInc<Tin, Tout>(Tin, Tin, Tin, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>, out Site<int>)

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the first input meeting the `outTripCriteria`. The ramp stops prematurely when the trip criteria is met on all sites, or with the last input less or equal `inTo`. Additionally provides the index of the input step found.

```
public Site<Tin> LinearStopFromToInc<Tin, Tout>(Tin inFrom, Tin to, Tin inIncrement, Tin  
inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout, bool>  
outTripCriteria, out Site<int> tripIndex)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

to Tin

inIncrement Tin

The input increment value for every step.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

oneMeasurement [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

outTripCriteria [Func](#)<Tout, [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

tripIndex Site<[int](#)>

Output - contains the index of the input step found.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToInc<Tin, Tout>(Tin, Tin, Tin, Tin, Tin, Func<Tin, Site<Tout>>, Func<Tout, bool>, out Site<int>, out Site<Tout>)

Performs a linear search from **inFrom** by increasing with **inIncrement**. Executes **oneMeasurement** at each step including the start point. Determines the first input meeting the **outTripCriteria**. The ramp stops prematurely when the trip criteria is met on all sites, or with the last input less or equal **inTo**. Additionally provides the index and output value of the input step found.

```
public Site<Tin> LinearStopFromToInc<Tin, Tout>(Tin inFrom, Tin to, Tin inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Func<Tout, bool> outTripCriteria, out Site<int> tripIndex, out Site<Tout> tripOut)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

to Tin

inIncrement Tin

The input increment value for every step.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

oneMeasurement [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

outTripCriteria [Func](#)<Tout, [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

tripIndex Site<[int](#)>

Output - contains the index of the input step found.

tripOut Site<Tout>

Output - contains the output value of the input step found.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToInc<Tin, Tout>(Tin, Tin, Tin, Tin, Tin, Func<Tin, Site<Tout>>, Tout)

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or with the last input less or equal `inTo`.

```
public Site<Tin> LinearStopFromToInc<Tin, Tout>(Tin inFrom, Tin to, Tin inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`to` Tin

`inIncrement` Tin

The input increment value for every step.

`inOffset` Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

`inNotFoundResult` Tin

The return value for the case when the trip criteria was never found.

`oneMeasurement` [Func<Tin, Site<Tout>>](#)

The action to execute for every measurement.

`outTarget` Tout

The (numeric) target output value to be searched.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToInc<Tin, Tout>(Tin, Tin, Tin, Tin, Tin, Func<Tin, Site<Tout>>, Tout, out Site<int>)

Performs a linear search from `inFrom` by increasing with `inIncrement`. Executes `oneMeasurement` at each step including the start point. Determines the input resulting in an output closest to the (numeric) `outTarget`. The ramp stops prematurely when the target output is surpassed on all sites, or with the last input less or equal `inTo`. Additionally provides the index of the input step found.

```
public Site<Tin> LinearStopFromToInc<Tin, Tout>(Tin inFrom, Tin to, Tin inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget, out Site<int> closestIndex)
```

Parameters

`inFrom` Tin

The starting point of the linear input ramp.

`to` Tin

`inIncrement` Tin

The input increment value for every step.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

oneMeasurement [Func](#)<Tin, Site<Tout>>

The action to execute for every measurement.

outTarget Tout

The (numeric) target output value to be searched.

closestIndex Site<[int](#)>

Output - contains the index of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearStopFromToInc<Tin, Tout>(Tin, Tin, Tin, Tin, Tin, Func<Tin, Site<Tout>>, Tout, out Site<int>, out Site<Tout>)

Performs a linear search from **inFrom** by increasing with **inIncrement**. Executes **oneMeasurement** at each step including the start point. Determines the input resulting in an output closest to the (numeric) **outTarget**. The ramp stops prematurely when the target output is surpassed on all sites, or with the last input less or equal **inTo**. Additionally provides the index and output value of the input step found.

```
public Site<Tin> LinearStopFromToInc<Tin, Tout>(Tin inFrom, Tin to, Tin inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tin, Site<Tout>> oneMeasurement, Tout outTarget, out Site<int> closestIndex, out Site<Tout> closestOut)
```

Parameters

inFrom Tin

The starting point of the linear input ramp.

to Tin

inIncrement Tin

The input increment value for every step.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

oneMeasurement [Func<Tin, Site<Tout>>](#)

The action to execute for every measurement.

outTarget Tout

The (numeric) target output value to be searched.

closestIndex Site<int>

Output - contains the index of the input step found.

closestOut Site<Tout>

Output - contains the output value of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

Namespace Csra.TheLib.Execute

Classes

[Digital](#)

[ScanNetwork](#)

[Search](#)

Class Digital

Namespace: [Csra.TheLib.Execute](#)

Assembly: Csra.dll

```
public class Digital : ILib.IExecute.IDigital
```

Inheritance

[object](#) ← Digital

Implements

[ILib.IExecute.IDigital](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

ContinueToConditionalStop(PatternInfo, Action)

Continues a pattern to the next conditional stop and executed the action.

```
public void ContinueToConditionalStop(PatternInfo pattern, Action action)
```

Parameters

pattern [PatternInfo](#)

Pattern to be executed.

action [Action](#)

Action to be called at the stop.

ForcePatternHalt()

Stops the currently running non-threaded pattern.

```
public void ForcePatternHalt()
```

ForcePatternHalt(PatternInfo)

Stops the given pattern. Works for both threaded and non-threaded patterns.

```
public void ForcePatternHalt(PatternInfo patternInfo)
```

Parameters

patternInfo [PatternInfo](#)

Pattern to halt.

RunPattern(PatternInfo)

Starts the pattern burst for the given pattern and waits for it to complete. Equivalent to calling [Start Pattern\(PatternInfo\)](#) and [WaitPatternDone\(PatternInfo\)](#) in sequence.

```
public void RunPattern(PatternInfo patternInfo)
```

Parameters

patternInfo [PatternInfo](#)

Pattern to run.

RunPattern(SiteVariant)

Starts the pattern burst for the given SiteVariant and waits for it to complete. Equivalent to calling [Start Pattern\(PatternInfo\)](#) and [WaitPatternDone\(PatternInfo\)](#) in sequence.

```
public void RunPattern(SiteVariant sitePatterns)
```

Parameters

sitePatterns SiteVariant

Sites run pattern.

RunPatternConditionalStop(PatternInfo, int, Func<PatternInfo, int, List<PinSite<double>>>)

Runs a pattern executing the specified func action at each conditional stop.

```
public List<PinSite<double>>> RunPatternConditionalStop(PatternInfo pattern, int  
numberOfStops, Func<PatternInfo, int, List<PinSite<double>>> func)
```

Parameters

pattern [PatternInfo](#)

Pattern to be executed.

numberOfStops [int](#)

Number of stops in the pattern.

func [Func](#)<PatternInfo, int, List<PinSite<double>>>

Func action to be called at each stop.

Returns

[List](#)<PinSite<double>>>

Concatenated list of all the measurements taken at each stop.

StartPattern(PatternInfo)

Starts the pattern burst for the given pattern without waiting for it to complete.

```
public void StartPattern(PatternInfo patternInfo)
```

Parameters

patternInfo [PatternInfo](#)

Pattern to start.

StartPattern(SiteVariant)

Starts the pattern burst for the given SiteVariant without waiting for it to complete.

```
public void StartPattern(SiteVariant sitePatterns)
```

Parameters

sitePatterns SiteVariant

Sites to start pattern.

WaitPatternDone(PatternInfo)

Waits for the given pattern to complete execution before returning. Works for both threaded and non-threaded patterns.

```
public void WaitPatternDone(PatternInfo patternInfo)
```

Parameters

patternInfo [PatternInfo](#)

Pattern to wait for completion.

Class ScanNetwork

Namespace: [Csra.TheLib.Execute](#)

Assembly: Csra.dll

```
public class ScanNetwork : ILib.IExecute.IScanNetwork
```

Inheritance

[object](#) ← ScanNetwork

Implements

[ILib.IExecute.IScanNetwork](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

RunDiagnosis(ScanNetworkPatternInfo, ScanNetworkPatternResults, bool)

Runs diagnosis reburst on failed core instances, which is obtained from the ScanNetwork pattern results.

```
public void RunDiagnosis(ScanNetworkPatternInfo scanNetworkPattern,  
ScanNetworkPatternResults nonDiagnosisResults, bool concurrentDiagnosis = false)
```

Parameters

scanNetworkPattern [ScanNetworkPatternInfo](#)

The [ScanNetworkPatternInfo](#) Object that is associated with the ScanNetwork pattern(set).

nonDiagnosisResults [ScanNetworkPatternResults](#)

The acquired ScanNetwork pattern results which contains failed core list.

concurrentDiagnosis [bool](#)

Optional. Whether to perform diagnosis on multiple core instances concurrently per reburst.

RunPattern(ScanNetworkPatternInfo)

Runs the ScanNetwork pattern(set) and reburst if needed until all icl instances' pass/fail results are obtained.

```
public void RunPattern(ScanNetworkPatternInfo scanNetworkPattern)
```

Parameters

scanNetworkPattern [ScanNetworkPatternInfo](#)

The [ScanNetworkPatternInfo](#) Object that is associated with the ScanNetwork pattern(set).

Class Search

Namespace: [Csra.TheLib.Execute](#)

Assembly: Csra.dll

```
public class Search : ILib.IExecute.ISearch
```

Inheritance

[object](#) ← Search

Implements

[ILib.IExecute.ISearch](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

LinearFullProcess<Tin, Tout>(List<Site<Tout>>, Tin, Tin, Tin, Tin, Func<Tout, bool>)

Processes the measurements of a linear full search to find the device input condition satisfying the trip criteria on the output.

```
public Site<Tin> LinearFullProcess<Tin, Tout>(List<Site<Tout>> outValues, Tin inFrom, Tin  
inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tout, bool> outTripCriteria)
```

Parameters

outValues [List](#)<Site<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement `Tin`

The per-step increment of the linear input ramp.

inOffset `Tin`

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

outTripCriteria `Func<Tout, bool>`

A delegate indicating the output meets the condition required for the input value searched.

Returns

`Site<Tin>`

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

`LinearFullProcess<Tin, Tout>(List<Site<Tout>>, Tin, Tin, Tin, Tin, Func<Tout, bool>, out Site<int>)`

Processes the measurements of a linear full search to find the device input condition satisfying the trip criteria on the output. Additionally provides the index of the input step found.

```
public Site<Tin> LinearFullProcess<Tin, Tout>(List<Site<Tout>> outValues, Tin inFrom, Tin  
inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tout, bool> outTripCriteria, out  
Site<int> tripIndex)
```

Parameters

outValues [List](#)<Site<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

outTripCriteria [Func](#)<Tout, [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

tripIndex Site<[int](#)>

Output - contains the index of the input step found.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(List<Site<Tout>>, Tin, Tin, Tin, Tin, Func<Tout, bool>, out Site<int>, out Site<Tout>)

Processes the measurements of a linear full search to find the device input condition satisfying the trip criteria on the output. Additionally provides the index and output value of the input step found.

```
public Site<Tin> LinearFullProcess<Tin, Tout>(List<Site<Tout>> outValues, Tin inFrom, Tin  
inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tout, bool> outTripCriteria, out  
Site<int> tripIndex, out Site<Tout> tripOut)
```

Parameters

outValues [List](#)<Site<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

outTripCriteria [Func](#)<Tout, [bool](#)>

A delegate indicating the output meets the condition required for the input value searched.

tripIndex Site<[int](#)>

Output - contains the index of the input step found.

tripOut Site<Tout>

Output - contains the output value of the input step found.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(List<Site<Tout>>, Tin, Tin, Tin, Tout)

Processes the measurements of a linear full search to find the device input condition resulting in an output closest to the (numeric) target.

```
public Site<Tin> LinearFullProcess<Tin, Tout>(List<Site<Tout>> outValues, Tin inFrom, Tin inIncrement, Tin inOffset, Tout outTarget)
```

Parameters

outValues [List](#)<Site<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

outTarget `Tout`

The (numeric) target output value to be searched.

Returns

`Site<Tin>`

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(List<Site<Tout>>, Tin, Tin, Tin, Tout, out Site<int>)

Processes the measurements of a linear full search to find the device input condition resulting in an output closest to the (numeric) target. Additionally provides the index of the input step found.

```
public Site<Tin> LinearFullProcess<Tin, Tout>(List<Site<Tout>> outValues, Tin inFrom, Tin  
inIncrement, Tin inOffset, Tout outTarget, out Site<int> closestIndex)
```

Parameters

outValues `List<Site<Tout>>`

The collected measurements for all executed steps.

inFrom `Tin`

The starting value of the linear input ramp.

inIncrement `Tin`

The per-step increment of the linear input ramp.

inOffset `Tin`

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

outTarget `Tout`

The (numeric) target output value to be searched.

closestIndex `Site<int>`

Output - contains the index of the input step found.

Returns

`Site<Tin>`

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(List<Site<Tout>>, Tin, Tin, Tin, Tout, out Site<int>, out Site<Tout>)

Processes the measurements of a linear full search to find the device input condition resulting in an output closest to the (numeric) target. Additionally provides the index and output value of the input step found.

```
public Site<Tin> LinearFullProcess<Tin, Tout>(List<Site<Tout>> outValues, Tin inFrom, Tin inIncrement, Tin inOffset, Tout outTarget, out Site<int> closestIndex, out Site<Tout> closestOut)
```

Parameters

outValues [List](#)<Site<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

outTarget Tout

The (numeric) target output value to be searched.

closestIndex Site<[int](#)>

Output - contains the index of the input step found.

closestOut Site<Tout>

Output - contains the output value of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(Site<Samples<Tout>>, Tin, Tin, Tin, Func<Tout, bool>)

Processes the measurements of a linear full search to find the device input condition satisfying the trip criteria on the output.

```
public Site<Tin> LinearFullProcess<Tin, Tout>(Site<Samples<Tout>> outValues, Tin inFrom, Tin inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tout, bool> outTripCriteria)
```

Parameters

outValues Site<Samples<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

outTripCriteria Func<Tout, bool>

A delegate indicating the output meets the condition required for the input value searched.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(Site<Samples<Tout>>, Tin, Tin, Tin, Func<Tout, bool>, out Site<int>)

Processes the measurements of a linear full search to find the device input condition satisfying the trip criteria on the output. Additionally provides the index of the input step found.

```
public Site<Tin> LinearFullProcess<Tin, Tout>(Site<Samples<Tout>> outValues, Tin inFrom, Tin  
inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tout, bool> outTripCriteria, out  
Site<int> tripIndex)
```

Parameters

outValues Site<Samples<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult Tin

The return value for the case when the trip criteria was never found.

outTripCriteria Func<Tout, bool>

A delegate indicating the output meets the condition required for the input value searched.

tripIndex Site<int>

Output - contains the index of the input step found.

Returns

Site<Tin>

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(Site<Samples<Tout>>, Tin, Tin, Tin, Func<Tout, bool>, out Site<int>, out Site<Tout>)

Processes the measurements of a linear full search to find the device input condition satisfying the trip criteria on the output. Additionally provides the index and output value of the input step found.

```
public Site<Tin> LinearFullProcess<Tin, Tout>(Site<Samples<Tout>> outValues, Tin inFrom, Tin inIncrement, Tin inOffset, Tin inNotFoundResult, Func<Tout, bool> outTripCriteria, out Site<int> tripIndex, out Site<Tout> tripOut)
```

Parameters

outValues Site<Samples<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement `Tin`

The per-step increment of the linear input ramp.

inOffset `Tin`

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

inNotFoundResult `Tin`

The return value for the case when the trip criteria was never found.

outTripCriteria `Func<Tout, bool>`

A delegate indicating the output meets the condition required for the input value searched.

tripIndex `Site<int>`

Output - contains the index of the input step found.

tripOut `Site<Tout>`

Output - contains the output value of the input step found.

Returns

`Site<Tin>`

The first input value resulting in an output satisfying the trip criteria.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(Site<Samples<Tout>>, Tin, Tin, Tin, Tout)

Processes the measurements of a linear full search to find the device input condition resulting in an output closest to the (numeric) target.

```
public Site<Tin> LinearFullProcess<Tin, Tout>(Site<Samples<Tout>> outValues, Tin inFrom, Tin inIncrement, Tin inOffset, Tout outTarget)
```

Parameters

outValues Site<Samples<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

outTarget Tout

The (numeric) target output value to be searched.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(Site<Samples<Tout>>, Tin, Tin, Tin, Tout, out Site<int>)

Processes the measurements of a linear full search to find the device input condition resulting in an output closest to the (numeric) target. Additionally provides the index of the input step found.

```
public Site<Tin> LinearFullProcess<Tin, Tout>(Site<Samples<Tout>> outValues, Tin inFrom, Tin inIncrement, Tin inOffset, Tout outTarget, out Site<int> closestIndex)
```

Parameters

outValues Site<Samples<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

outTarget Tout

The (numeric) target output value to be searched.

closestIndex Site<int>

Output - contains the index of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

LinearFullProcess<Tin, Tout>(Site<Samples<Tout>>, Tin, Tin, Tin, Tout, out Site<int>, out Site<Tout>)

Processes the measurements of a linear full search to find the device input condition resulting in an output closest to the (numeric) target. Additionally provides the index and output value of the input step found.

```
public Site<Tin> LinearFullProcess<Tin, Tout>(Site<Samples<Tout>> outValues, Tin  
inFrom, Tin inIncrement, Tin inOffset, Tout outTarget, out Site<int> closestIndex, out  
Site<Tout> closestOut)
```

Parameters

outValues Site<Samples<Tout>>

The collected measurements for all executed steps.

inFrom Tin

The starting value of the linear input ramp.

inIncrement Tin

The per-step increment of the linear input ramp.

inOffset Tin

The offset to correct the calculated input value. Use negative values to compensate propagation delays for output switching.

outTarget Tout

The (numeric) target output value to be searched.

closestIndex Site<int>

Output - contains the index of the input step found.

closestOut Site<Tout>

Output - contains the output value of the input step found.

Returns

Site<Tin>

The input value resulting in an output closest to the target.

Type Parameters

Tin

The type of the input condition for the device.

Tout

The type of the device's output.

Namespace Csra.TheLib.Setup

Classes

[Dc](#)

[Digital](#)

[LevelsAndTiming](#)

Class Dc

Namespace: [Csra.TheLib.Setup](#)

Assembly: Csra.dll

```
public class Dc : ILib.ISetup.IDc
```

Inheritance

[object](#) ← Dc

Implements

[ILib.ISetup.IDc](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Extension Methods

[CustomerExtensions.CustomerExtension\(ILib.ISetup.IDc, string, int\)](#).

Methods

Connect(Pins, bool)

Connects and optionally gates on/off the pins depending on its instrument feature (PPMU, DCVI, DCVS,...).

```
public void Connect(Pins pins, bool gateOn = false)
```

Parameters

pins [Pins](#)

The pins to connect.

gateOn [bool](#)

Optional. Default no gate change, True for gate on the pins after connecting.

ConnectAllPins()

Connects all power and digital pins from level context.

```
public void ConnectAllPins()
```

Disconnect(Pins, bool)

Disconnects and optionally gates on/off the pins depending on its instrument feature (PPMU, DCVI, DCVS,...). It will disconnect in HiZ mode rather than forcing 0V or 0A on VIs.

```
public void Disconnect(Pins pins, bool gateOff = true)
```

Parameters

pins [Pins](#)

The pins to disconnect.

gateOff [bool](#) ↗

Optional. Default gate off (HiZ) the pins before disconnecting, False no gate change.

Force(Pins, TLibOutputMode, double, double, double, bool)

Sets the force current, force voltage or high impedance of the pins.

```
public void Force(Pins pins, TLibOutputMode mode, double forceValue, double forceRange,
double clampValue, bool gateOn = true)
```

Parameters

pins [Pins](#)

The pins to force.

mode [TLibOutputMode](#)

The mode for forcing (e.g., Voltage or Current).

forceValue [double](#)

The value to force.

forceRange [double](#)

The range for force value.

clampValue [double](#)

When forcing Voltage it sets the current limit and when forcing Current it sets the voltage range.

gateOn [bool](#)

Optional. Default gate on the pins after after the settings, False no gate change.

Force(Pins[], TLibOutputMode[], double[], double[], double[], bool[])

Sets the force current, force voltage or high impedance of each element in pinGroups.

```
public void Force(Pins[] pinGroups, TLibOutputMode[] modes, double[] forceValues, double[] forceRanges, double[] clampValues, bool[] gateOn = null)
```

Parameters

pinGroups [Pins](#)[]

Array of pin or pin groups.

modes [TLibOutputMode](#)[]

Array of the mode for each pin or pin group.

forceValues [double](#)[]

Array of force values for each pin or pin group.

forceRanges [double](#)[]

Array of force ranges for each pin or pin group.

clampValues [double](#)[]

Array of clamp values for each pin or pin group.

gateOn [bool](#)[]

Optional. Array of gate state for each pin or pin group, default gate on for all pin or pin group.

ForceHiZ(Pins)

Sets to High Impedance mode.

```
public void ForceHiZ(Pins pins)
```

Parameters

pins [Pins](#)

The pins to set in HiZ.

ForceI(Pins, double, double, double?, bool, bool, double?)

Sets the Force Current and the range of a DC instrument. Assuming that the instrument was already setup for remaining parameters.

```
public void ForceI(Pins pins, double forceCurrent, double clampHiV, double currentRange, double? voltageRange = null, bool outputModeCurrent = false, bool gateOn = true, double? clampLoV = null)
```

Parameters

pins [Pins](#)

The pins to force the current.

forceCurrent [double](#)[]

The current to force.

clampHiV [double](#)[]

Sets the voltage for voltage clamp high.

currentRange [double](#)?

Expected current to set the current range.

voltageRange [double](#)?

Optional. Expected voltage to set the voltage range or to program the voltage for DCVS.

outputModeCurrent [bool](#)

Optional. Sets to true to switch to force current mode (if the mode was not previously set).

gateOn [bool](#)

Optional. Default gate on the pins after after the settings, False no gate change.

clampLoV [double](#)?

Optional. Sets the voltage for voltage clamp low.

ForceI(Pins, double, double?, double?, bool, bool)

Sets the Force current of a DC instrument. Assumes the instrument was already setup to the right modes.

```
public void ForceI(Pins pins, double forceCurrent, double? clampHiV = null, double? clampLoV = null, bool outputModeCurrent = false, bool gateOn = true)
```

Parameters

pins [Pins](#)

The pins to force the current.

forceCurrent [double](#)?

The current to force.

clampHiV [double](#)?

Optional. Sets the voltage for voltage clamp high.

clampLoV [double](#)?

Optional. Sets the voltage for voltage clamp low.

outputModeCurrent [bool](#)

Optional. Sets to true to switch to force current mode (if the mode was not previously set).

gateOn [bool](#)

Optional. Default gate on the pins after after the settings, False no gate change.

ForceV(Pins, double, double, double?, bool, bool)

Programs the force Voltage, measure range and many other parameters of a DC instrument - Advanced method with additional parameters.

```
public void ForceV(Pins pins, double forceVoltage, double clampCurrent, double voltageRange, double? currentRange = null, bool outputModeVoltage = false, bool gateOn = true)
```

Parameters

pins [Pins](#)

The pins to force the voltage.

forceVoltage [double](#)

The force voltage value.

clampCurrent [double](#)

Current clamp value.

voltageRange [double](#)

Expected voltage to set the Voltage range, if required, else use the forceVoltage.

currentRange [double](#)?

Optional. Expected current to set the current range.

outputModeVoltage [bool](#)

Optional. Sets to true to switch to force voltage mode (if the mode was not previously set).

gateOn [bool](#)

Optional. Default gate on the pins after after the settings, False no gate change.

ForceV(Pins, double, double?, bool, bool)

Programs the force Voltage of a DC instrument. Simplest Method: It assumes the instrument is already in the right mode (FI,FV) and required ranges.

```
public void ForceV(Pins pins, double forceVoltage, double? clampCurrent = null, bool  
outputModeVoltage = false, bool gateOn = true)
```

Parameters

pins [Pins](#)

The pins to force the voltage.

forceVoltage [double](#)

The force voltage that will be set.

clampCurrent [double](#)?

Optional. Current clamp value.

outputModeVoltage [bool](#)

Optional. Sets to true to switch to force voltage mode (if the mode was not previously set).

gateOn [bool](#)

Optional. Default gate on the pins after after the settings, False no gate change.

Modify(Pins, DcParameters)

Selectively program or modify any DC instrument parameter.

```
public void Modify(Pins pins, DcParameters parameters)
```

Parameters

pins [Pins](#)

The pins to set.

parameters [DcParameters](#)

The object through which each parameter can be set.

Modify(Pins, bool?, TLibOutputMode?, double?, double?, double?, double?, double?, double?, double?, Measure?, double?, bool?, bool?, double?, double?, double?, double?, double?, double?, double?, double?, double?, double?)

Selectively program or modify any DC instrument parameter.

```
public void Modify(Pins pins, bool? gate = null, TLibOutputMode? mode = null, double?  
voltage = null, double? voltageAlt = null, double? current = null, double? voltageRange =  
null, double? currentRange = null, double? forceBandwidth = null, Measure? meterMode = null,  
double? meterVoltageRange = null, double? meterCurrentRange = null, double? meterBandwidth =  
null, double? sourceFoldLimit = null, double? sinkFoldLimit = null, double?  
sourceOverloadLimit = null, double? sinkOverloadLimit = null, bool? voltageAltOutput = null,  
bool? bleederResistor = null, double? complianceBoth = null, double? compliancePositive =  
null, double? complianceNegative = null, double? clampHiV = null, double? clampLoV = null,  
bool? highAccuracy = null, double? settlingTime = null, double? hardwareAverage = null)
```

Parameters

pins [Pins](#)

The pins to set.

gate [bool](#)?

Optional. Sets the gate.

mode [TLibOutputMode](#)?

Optional. Sets the operating mode.

voltage [double](#)?

Optional. Sets the output voltage.

voltageAlt [double](#)?

Optional. Sets the alternate output voltage.

current [double](#)?

Optional. Sets the output current.

voltageRange [double](#)?

Optional. Sets the voltage range.

currentRange [double](#)?

Optional. Sets the current range.

forceBandwidth [double](#)?

Optional. Sets the output compensation bandwidth.

meterMode [Measure](#)?

Optional. Sets the meter mode.

meterVoltageRange [double](#)?

Optional. Sets the meter voltage range.

meterCurrentRange [double](#)?

Optional. Sets the meter current range.

meterBandwidth [double](#)?

Optional. Sets the meter filter.

sourceFoldLimit [double](#)?

Optional. Sets the source fold limit.

sinkFoldLimit [double](#)?

Optional. Sets the sink fold limit.

sourceOverloadLimit [double](#)?

Optional. Sets the source overload limit.

sinkOverloadLimit [double](#)?

Optional. Sets the sink overload limit.

voltageAltOutput [bool](#)?

Optional. Sets the output DAC used to force voltage (true for alternate or false for main).

bleederResistor [bool](#)?

Optional. Sets the bleeder resistor's connection state.

complianceBoth [double](#)?

Optional. Sets both compliance ranges.

compliancePositive [double](#)?

Optional. Sets the positive compliance range.

complianceNegative [double](#)?

Optional. Sets the negative compliance range.

clampHiV [double](#)?

Optional. Sets the high voltage clamp value.

clampLoV [double](#)?

Optional. Sets the low voltage clamp value.

highAccuracy [bool](#)?

Optional. Sets the enabled state of the high accuracy measure voltage.

settlingTime [double](#)?

Optional. Sets the required additional settling time for the high accuracy measure voltage mode.

hardwareAverage [double](#)?

Optional. Sets the meter hardware average value.

SetForceAndMeter(Pins, TLibOutputMode, double, double, double, Measure, double, bool)

Programs the force and the meter's measure parameters interface for the dc instruments.

```
public void SetForceAndMeter(Pins pins, TLibOutputMode mode, double forceValue, double forceRange, double clampValue, Measure meterMode, double measureRange, bool gateOn = true)
```

Parameters

pins [Pins](#)

The pins to set force and meter parameters.

mode [TLibOutputMode](#)

Set the output mode to TlibOutputMode Voltage, Current or HiZ.

forceValue [double](#)

Force voltage or current value.

forceRange [double](#)

Voltage or current to set the force range.

clampValue [double](#)

Current or voltage clamp value. Note: For PPMU it programs either clampVHi or clampVLo depending if sourcing or sinking current.

meterMode [Measure](#)

Set the meter's measure mode to measure voltage or current.

measureRange [double](#)

Set the meter's measure range to the expected current or voltage.

gateOn [bool](#)

Optional. Default gate on the pins after after the settings, False no gate change.

SetMeter(Pins, Measure, double, double?, int?, double?)

Sets the measurement interface of the instruments DCVI and DCSV.

```
public void SetMeter(Pins pins, Measure meterMode, double rangeValue, double? filterValue = null, int? hardwareAverage = null, double? outputRangeValue = null)
```

Parameters

pins [Pins](#)

The pins to set meter parameters.

meterMode [Measure](#)

Set the mode to measure Voltage or Current.

rangeValue [double](#)?

Current or Voltage range depending on the selected mode.

filterValue [double](#)?

Optional. Sets the filter value.

hardwareAverage [int](#)?

Optional. Sets the hardware average for the specified DCVI pins.

outputRangeValue [double](#)?

Optional. Current range for DCSV when you want to set the current mode - for other cases, ignore this.

SetMeter(Pins[], Measure[], double[], double[], int[], double[])

Sets the measurements interface of the instruments DCVI and DCSV.

```
public void SetMeter(Pins[] pinGroups, Measure[] meterModes, double[] rangeValues, double[] filterValues = null, int[] hardwareAverages = null, double[] outputRangeValues = null)
```

Parameters

pinGroups [Pins\[\]](#)

Array of pin or pin groups.

meterModes [Measure\[\]](#)

Array of settings measurements mode voltage and current.

rangeValues [double\[\]](#)[]

Array of current and voltage range depending on the selected modes.

filterValues [double\[\]](#)[]

Optional. Array of filter values.

hardwareAverages [int\[\]](#)[]

Optional. Array of hardware average for the specified DCVI pins.

outputRangeValues [double\[\]](#)[]

Optional. Array of current range for DCSV when you want to set the current mode - for other cases, ignore this.

Class Digital

Namespace: [Csra.TheLib.Setup](#)

Assembly: Csra.dll

```
public class Digital : ILib.ISetup.IDigital
```

Inheritance

[object](#) ← Digital

Implements

[ILib.ISetup.IDigital](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Connect(Pins)

Connect digital pins to the digital driver and comparator

```
public void Connect(Pins pins)
```

Parameters

pins [Pins](#)

Pins to be connected, must contain digital pins

Disconnect(Pins)

Disconnect digital pins from the digital driver and comparator

```
public void Disconnect(Pins pins)
```

Parameters

pins [Pins](#)

Pins to be disconnected, must contain digital pins

FrequencyCounter(Pins, double, FreqCtrEventSrcSel, FreqCtrEventSlopeSel)

Configures the digital instrument frequency counter.

```
public void FrequencyCounter(Pins pins, double measureWindow, FreqCtrEventSrcSel eventSource, FreqCtrEventSlopeSel eventSlope)
```

Parameters

pins [Pins](#)

Digital pin(s) to be measured.

measureWindow [double](#) ↗

Time to measure the frequency.

eventSource FreqCtrEventSrcSel

The frequency counters comparator threshold.

eventSlope FreqCtrEventSlopeSel

The frequency counters event slope.

ModifyPins(Pins, DigitalPinsParameters)

Selectively program or modify any digital instrument pins parameter.

```
public void ModifyPins(Pins pins, DigitalPinsParameters parameters)
```

Parameters

pins [Pins](#)

The pins to set.

parameters [DigitalPinsParameters](#)

The object through which each parameter can be set.

ModifyPins(Pins, tlHSDMAlarm?, tlAlarmBehavior?, bool?, bool?, ChInitState?, ChStartState?, bool?, bool?)

Selectively program or modify any digital instrument Pins parameter.

```
public void ModifyPins(Pins pins, tlHSDMAlarm? alarmType = null, tlAlarmBehavior?
alarmBehavior = null, bool? disableCompare = null, bool? disableDrive = null, ChInitState?
initState = null, ChStartState? startState = null, bool? calibrationExcluded = null, bool?
calibrationHighAccuracy = null)
```

Parameters

pins [Pins](#)

The pins to set.

alarmType [tlHSDMAlarm?](#)

Optional. Sets the alarm type for the specified pins.

alarmBehavior [tlAlarmBehavior?](#)

Optional. Sets the alarm behavior for the specified pins.

disableCompare [bool](#)?

Optional. Disables the comparators for the specified pins

disableDrive [bool](#)?

Optional. Disables the drivers for the specified pins

initState [ChInitState?](#)

Optional. Sets the initial state of the pins

startState ChStartState?

Optional. Sets the start state of the pins

calibrationExcluded [bool](#)?

Optional. Sets the specified pins to be excluded from job dependent calibration

calibrationHighAccuracy [bool](#)?

Optional. Enables or disables calibration high accuracy mode for the specified pins

ModifyPinsLevels(Pins, DigitalPinsLevelsParameters)

Selectively program or modify any digital instrument pins levels parameter.

```
public void ModifyPinsLevels(Pins pins, DigitalPinsLevelsParameters parameters)
```

Parameters

pins [Pins](#)

The pins to set.

parameters [DigitalPinsLevelsParameters](#)

The object through which each parameter can be set.

ModifyPinsLevels(Pins, ChDiffPinLevel?, double?, TLibDiffLvlValType[], double[], tlDriverMode?, ChPinLevel?, double?, SiteDouble, PinListData)

Selectively program or modify any digital instrument pins levels parameter.

```
public void ModifyPinsLevels(Pins pins, ChDiffPinLevel? differentialLevelsType = null,
double? differentialLevelsValue = null, TLibDiffLvlValType[] differentialLevelsValuesType =
null, double[] differentialLevelsValues = null, tlDriverMode? levelsDriverMode = null,
```

```
ChPinLevel? levelsType = null, double? levelsValue = null, SiteDouble levelsValuePerSite = null, PinListData levelsValues = null)
```

Parameters

pins [Pins](#)

The pins to set.

differentialLevelsType ChDiffPinLevel?

Optional. Sets the differential levels type for the specified pins

differentialLevelsValue [double](#)?

Optional. Sets the specified differential pin level type for the specified pins

differentialLevelsValuesType [TLibDiffLvlValType](#)[]

Optional. Sets the differential levels values type for the specified pins

differentialLevelsValues [double](#)[][]

Optional. Sets the specified differential pin levels values type for the specified pins

levelsDriverMode tIDriverMode?

Optional. Sets the driver mode for the specified pins

levelsType ChPinLevel?

Optional. Sets the level type for the specified pins

levelsValue [double](#)?

Optional. Sets the value for the specified level type on the specified pins

levelsValuePerSite SiteDouble

Optional. Sets the value for the specified level type for the specified pins on each site

levelsValues PinListData

Optional. Sets the value for the specified level value for each specified site and each specified pin

ModifyPinsTiming(Pins, DigitalPinsTimingParameters)

Selectively program or modify any digital instrument pins timing parameter.

```
public void ModifyPinsTiming(Pins pins, DigitalPinsTimingParameters parameters)
```

Parameters

pins [Pins](#)

The pins to set.

parameters [DigitalPinsTimingParameters](#)

The object through which each parameter can be set.

```
ModifyPinsTiming(Pins, double?, double?, bool?, string, chEdge?, bool?, double?, double?, string, double?, tlOffsetType?, double?, bool?, SiteLong, int?, SiteDouble, AutoStrobeEnableSel?, int?, int?, double?, double?, bool?, double?, FreqCtrEnableSel?, FreqCtrEventSlopeSel?, FreqCtrEventSrcSel?, double?)
```

Selectively program or modify any digital instrument pins timing parameter.

```
public void ModifyPinsTiming(Pins pins, double? timingClockOffset = null, double? timingClockPeriod = null, bool? timingDisableAllEdges = null, string timingEdgeSet = null, chEdge? timingEdgeVal = null, bool? timingEdgeEnabled = null, double? timingEdgeTime = null, double? timingRefOffset = null, string timingSetup1xDiagnosticCapture = null, double? timingSrcSyncDataDelay = null, tlOffsetType? timingOffsetType = null, double? timingOffsetValue = null, bool? timingOffsetEnabled = null, SiteLong timingOffsetSelectedPerSite = null, int? timingOffsetValuePerSiteIndex = null, SiteDouble timingOffsetValuePerSiteValue = null, AutoStrobeEnableSel? autoStrobeEnabled = null, int? autoStrobeNumSteps = null, int? autoStrobeSamplesPerStep = null, double? autoStrobeStartTime = null, double? autoStrobeStepTime = null, bool? freeRunningClockEnabled = null, double? freeRunningClockFrequency = null, FreqCtrEnableSel? freqCtrEnable = null, FreqCtrEventSlopeSel? freqCtrEventSlope = null, FreqCtrEventSrcSel? freqCtrEventSource = null, double? freqCtrInterval = null)
```

Parameters

pins [Pins](#)

The pins to set.

timingClockOffset [double](#)?

Optional. Sets the offset value between a DQS bus and a DUT clock in a DDR Protocol Aware test program for the specified pins

timingClockPeriod [double](#)?

Optional. Sets the current value for the period for the specified clock pins

timingDisableAllEdges [bool](#)?

Optional. Disables all edges (drive and compare) for the specified pins

timingEdgeSet [string](#)?

Optional. Sets the edgeset name for the specified pins

timingEdgeVal chEdge?

Optional. Sets the timing edge for the specified pins

timingEdgeEnabled [bool](#)?

Optional. Sets the enabled state for the specified pins and timing edge

timingEdgeTime [double](#)?

Optional. Sets the edge value for the specified pins and timing edge

timingRefOffset [double](#)?

Optional. Sets the offset value between the specified source synchronous reference (clock) pin and its data pins

timingSetup1xDiagnosticCapture [string](#)?

Optional. Sets up special dual-bit diagnostic capture in CMEM fail capture (LFVM) memory using the 1X pin setup for the specified pins and Time Sets sheet name

timingSrcSyncDataDelay [double](#)?

Optional. Sets the strobe reference data delay for individual source synchronous data pins

timingOffsetType tIOffsetType?

Optional. Sets the timing offset type for the specified pins

timingOffsetValue [double](#)?

Optional. Sets the timing offset value for the specified pins

timingOffsetEnabled [bool](#)?

Optional. Sets the timing offset enabled state for the specified pins

timingOffsetSelectedPerSite SiteLong

Optional. Sets the active offset index value for the specified pins on each site

timingOffsetValuePerSiteIndex [int](#)?

Optional. Set the timing offset index value. The valid index range is 0-7

timingOffsetValuePerSiteValue SiteDouble

Optional. Sets the current value for the offset at a specific index location that is to be applied to the timing values for the specified pins on each site

autoStrobeEnabled AutoStrobeEnableSel?

Optional. Enable state of the AutoStrobe engine for the specified pins

autoStrobeNumSteps [int](#)?

Optional. Sets the number of steps on the AutoStrobe engines for the specified pins

autoStrobeSamplesPerStep [int](#)?

Optional. Sets the number of samples per step on the AutoStrobe engines for the specified pins

autoStrobeStartTime [double](#)?

Optional. Sets the start time on the AutoStrobe engines for the specified pins

autoStrobeStepTime [double](#)?

Optional. Sets the step time on the AutoStrobe engines for the specified pins

freeRunningClockEnabled [bool](#)?

Optional. Sets the enable state of the free-running clock for the specified pins

freeRunningClockFrequency [double](#)?

Optional. Sets the frequency of the free-running clock for the specified pins

freqCtrEnable FreqCtrEnableSel?

Optional. Sets the frequency counter's enable state for the specified pins

freqCtrEventSlope FreqCtrEventSlopeSel?

Optional. Sets the frequency counter's event slope for the specified pins

freqCtrEventSource FreqCtrEventSrcSel?

Optional. Sets the frequency counter's event source for the specified pins

freqCtrInterval [double](#)?

Optional. Sets the duration of time to capture the frequency counter data for the specified pins

ReadAll()

Configures the tester to read all the vector data using HRAM.

```
public void ReadAll()
```

ReadFails()

Configures the tester to read the failing vector data using HRAM.

```
public void ReadFails()
```

ReadHram(int, CaptType, TrigType, bool, int)

Configures the tester for HRAM read back.

```
public void ReadHram(int captureLimit, CaptType captureType, TrigType triggerType, bool
waitForEvent, int preTriggerCycleCount)
```

Parameters

captureLimit int↗

Maximum number of vectors to be captured

captureType CaptType

Cycle type to be captured by HRAM, options are all, fail or stv

triggerType TrigType

Type of trigger for capture cycles, options are fail, first or never

waitForEvent bool↗

Sets whether the trigger waits for a cycle, vector or loop event

preTriggerCycleCount int↗

The number of cycles to capture before the trigger cycle.

ReadStoredVectors()

Configures the tester to read the data from vectors containing the STV statement.

```
public void ReadStoredVectors()
```

Class LevelsAndTiming

Namespace: [Csra.TheLib.Setup](#)

Assembly: Csra.dll

```
public class LevelsAndTiming : ILib.ISetup.ILevelsAndTiming
```

Inheritance

[object](#) ← LevelsAndTiming

Implements

[ILib.ISetup.ILevelsAndTiming](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Apply(bool, bool, bool)

Apply Connections, Levels and Timing, in either powered or unpowered mode.

```
public void Apply(bool connectAllPins = false, bool unpowered = false, bool  
levelRampSequence = false)
```

Parameters

connectAllPins [bool](#)

Optional. If true: Connect all pins.

unpowered [bool](#)

Optional. If true: power down instruments and power supplies before connecting all pins.

levelRampSequence [bool](#)

Optional. If true: will ramp all levels with a predefined slew rate and sequence.

ApplyWithPinStates(bool, bool, bool, Pins, Pins, Pins)

Apply Connections, Level, Timing and set init states, in either powered or unpowered mode.

```
public void ApplyWithPinStates(bool connectAllPins = false, bool unpowered = false, bool levelRampSequence = false, Pins initPinsHi = null, Pins initPinsLo = null, Pins initPinsHiZ = null)
```

Parameters

`connectAllPins` [bool](#)

Optional. If true: Connect all pins.

`unpowered` [bool](#)

Optional. If true: power down instruments and power supplies before connecting all pins.

`levelRampSequence` [bool](#)

Optional. If true: will ramp all levels with a predefined slew rate and sequence.

`initPinsHi` [Pins](#)

Optional. Pin or pingroup to initialize to drive state high.

`initPinsLo` [Pins](#)

Optional. Pin or pingroup to initialize to drive state low.

`initPinsHiZ` [Pins](#)

Optional. Pin or pingroup to initialize to drive state tri-state.

Class Datalog

Namespace: [Csra.TheLib](#)

Assembly: Csra.dll

```
public class Datalog : ILib.IDatalog
```

Inheritance

[object](#) ← Datalog

Implements

[ILib.IDatalog](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

TestFunctional(Site<bool>, string)

Perform a functional datalog test.

```
public void TestFunctional(Site<bool> result, string pattern = "")
```

Parameters

result Site<[bool](#)>

The result object to be datalogged.

pattern string

Optional. The pattern executed.

TestParametric(PinSite<double>, double, string)

Perform a parametric datalog test by using FlowLimits.

```
public void TestParametric(PinSite<double> result, double forceValue = 0, string forceUnit  
= "")
```

Parameters

result PinSite<[double](#)>

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

TestParametric(PinSite<int>, double, string)

Perform a parametric datalog test by using FlowLimits.

```
public void TestParametric(PinSite<int> result, double forceValue = 0, string forceUnit  
= "")
```

Parameters

result PinSite<[int](#)>

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

TestParametric(PinSite<Samples<double>>, double, string, bool)

Perform a parametric datalog test by using FlowLimits.

```
public void TestParametric(PinSite<Samples<double>> result, double forceValue = 0, string  
forceUnit = "", bool sameLimitForAllSamples = false)
```

Parameters

result PinSite<Samples<[double](#)>>

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

sameLimitForAllSamples [bool](#)

Optional. Whether to use the same FlowLimit for all samples.

TestParametric(PinSite<Samples<int>>, double, string, bool)

Perform a parametric datalog test by using FlowLimits.

```
public void TestParametric(PinSite<Samples<int>> result, double forceValue = 0, string  
forceUnit = "", bool sameLimitForAllSamples = false)
```

Parameters

result PinSite<Samples<[int](#)>>

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

sameLimitForAllSamples [bool](#)

Optional. Whether to use the same FlowLimit for all samples.

TestParametric(Site<double>, double, string)

Perform a parametric datalog test by using FlowLimits.

```
public void TestParametric(Site<double> result, double forceValue = 0, string forceUnit  
= "")
```

Parameters

result Site<[double](#)>

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

TestParametric(Site<int>, double, string)

Perform a parametric datalog test by using FlowLimits.

```
public void TestParametric(Site<int> result, double forceValue = 0, string forceUnit = "")
```

Parameters

result Site<[int](#)>

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

TestParametric(Site<Samples<double>>, double, string, bool)

Perform a parametric datalog test by using FlowLimits.

```
public void TestParametric(Site<Samples<double>> result, double forceValue = 0, string  
forceUnit = "", bool sameLimitForAllSamples = false)
```

Parameters

result Site<Samples<[double](#)>>

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

sameLimitForAllSamples [bool](#)

Optional. Whether to use the same FlowLimit for all samples.

TestParametric(Site<Samples<int>>, double, string, bool)

Perform a parametric datalog test by using FlowLimits.

```
public void TestParametric(Site<Samples<int>> result, double forceValue = 0, string  
forceUnit = "", bool sameLimitForAllSamples = false)
```

Parameters

result Site<Samples<[int](#)>>

The result object to be datalogged.

forceValue [double](#)

Optional. The force value applied for the result.

forceUnit [string](#)

Optional. The force value's unit.

sameLimitForAllSamples [bool](#)

Optional. Whether to use the same FlowLimit for all samples.

TestScanNetwork(ScanNetworkPatternResults, ScanNetworkDatalogOption)

Perform a flexible datalog test for ScanNetwork pattern results, with datalogging options set by [ScanNetworkDatalogOption](#).

```
public void TestScanNetwork(ScanNetworkPatternResults result, ScanNetworkDatalogOption  
datalogOptions)
```

Parameters

result [ScanNetworkPatternResults](#)

The ScanNetwork pattern result object of type [ScanNetworkPatternResults](#)

datalogOptions [ScanNetworkDatalogOption](#)

Class Validate

Namespace: [Csra.TheLib](#)

Assembly: Csra.dll

```
public class Validate : ILib.IValidate
```

Inheritance

[object](#) ← Validate

Implements

[ILib.IValidate](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Dc(Pins, bool?, TLibOutputMode?, double?, double?, double?,
double?, double?, double?, Measure?, double?, double?,
double?, double?, double?, double?, bool?, bool?,
double?, double?, double?, double?, bool?, double?,
double?)

Validate arguments within Test Method Templates.

```
public void Dc(Pins pins, bool? gate = null, TLibOutputMode? mode = null, double? voltage = null, double? voltageAlt = null, double? current = null, double? voltageRange = null, double? currentRange = null, double? forceBandwidth = null, Measure? meterMode = null, double? meterVoltageRange = null, double? meterCurrentRange = null, double? meterBandwidth = null, double? sourceFoldLimit = null, double? sinkFoldLimit = null, double? sourceOverloadLimit = null, double? sinkOverloadLimit = null, bool? voltageAltOutput = null, bool? bleederResistor = null, double? complianceBoth = null, double? compliancePositive = null, double? complianceNegative = null, double? clampHiV = null, double? clampLoV = null, bool? highAccuracy = null, double? settlingTime = null, double? hardwareAverage = null)
```

Parameters

pins [Pins](#)

Pins parameter to be validated.

gate [bool](#)?

Optional. Gate parameter to be validated.

mode [TLibOutputMode](#)?

Optional. Mode parameter to be validated.

voltage [double](#)?

Optional. Voltage parameter to be validated.

voltageAlt [double](#)?

Optional. VoltageAlt parameter to be validated.

current [double](#)?

Optional. Current parameter to be validated.

voltageRange [double](#)?

Optional. Voltage Range parameter to be validated.

currentRange [double](#)?

Optional. Current Range parameter to be validated.

forceBandwidth [double](#)?

Optional. Force Bandwidth parameter to be validated.

meterMode [Measure](#)?

Optional. Meter Mode parameter to be validated.

meterVoltageRange [double](#)?

Optional. Meter Voltage Range parameter to be validated.

meterCurrentRange [double](#)?

Optional. Meter Current Range parameter to be validated.

meterBandwidth [double](#)?

Optional. Meter Bandwidth parameter to be validated.

sourceFoldLimit [double](#)?

Optional. Source Fold Limit parameter to be validated.

sinkFoldLimit [double](#)?

Optional. Sink Fold Limit parameter to be validated.

sourceOverloadLimit [double](#)?

Optional. Source Overload Limit parameter to be validated.

sinkOverloadLimit [double](#)?

Optional. Sink Overload Limit parameter to be validated.

voltageAltOutput [bool](#)?

Optional. Voltage Alt Output parameter to be validated.

bleederResistor [bool](#)?

Optional. Bleeder Resistor parameter to be validated.

complianceBoth [double](#)?

Optional. Compliance Both parameter to be validated.

compliancePositive [double](#)?

Optional. Compliance Positive parameter to be validated.

complianceNegative [double](#)?

Optional. Compliance Negative parameter to be validated.

clampHiV [double](#)?

Optional. Clamp High V parameter to be validated.

clampLoV [double](#)?

Optional. Clamp Low V parameter to be validated.

highAccuracy [bool](#)?

Optional. High Accuracy parameter to be validated.

settlingTime [double](#)?

Optional. Settling Time parameter to be validated.

hardwareAverage [double](#)?

Optional. Hardware Average parameter to be validated.

Enum<T>(string, string, out T)

Checks if the provided string can be parsed to the specified enum type and create the enum value.

```
public bool Enum<T>(string value, string argumentName, out T enumValue) where T :  
    struct, Enum
```

Parameters

value [string](#)

The string to parse for. Case-Insensitive, specify only the enum member part.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

enumValue [T](#)

The enumeration value to output in the event that the provided value was successfully parsed in the provided enumeration.

Returns

[bool](#)

[true](#) if the **value** was found within the provided Enumeration and successfully parsed; otherwise, [false](#).

Type Parameters

T

The existing Enumeration to be parsed.

Fail(string, string)

Raises an unconditional validation error.

```
public void Fail(string problemReasonResolutionMessage, string argumentName)
```

Parameters

problemReasonResolutionMessage [string](#)

Validation failure message clearly describing the problem, reason, and resolution message.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

GreaterOrEqual<T>(T, T, string)

Checks if a numeric value is greater or equal to a bound.

```
public bool GreaterOrEqual<T>(T value, T boundary, string argumentName) where T : IComparable<T>
```

Parameters

value T

The value to be checked against the provided boundary.

boundary T

The boundary that the value must be greater than or equal to.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

Returns

[bool](#)

[true](#) if the **value** is greater than or equal to the provided boundary; otherwise, [false](#).

Type Parameters

T

The type specified for the provided parameters.

GreaterThan<T>(T, T, string)

Checks if a numeric value is greater than the provided boundary.

```
public bool GreaterThan<T>(T value, T boundary, string argumentName) where T : IComparable<T>
```

Parameters

value T

The value to be checked against the provided boundary.

boundary T

The boundary that the value must be greater than.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

Returns

[bool](#)

[true](#) if the 'value' is greater than the provided boundary; otherwise, [false](#).

Type Parameters

T

The type specified for the provided parameters.

InRange<T>(T, T, T, string)

Checks if a numeric value is within a range (including).

```
public bool InRange<T>(T value, T from, T to, string argumentName) where T : IComparable<T>
```

Parameters

value T

The value to be checked against the provided upper and lower threshold.

from T

The lower threshold.

to T

The upper threshold.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

Returns

[bool](#)

[true](#) if the **value** is within the provided range; otherwise, [false](#).

Type Parameters

T

The type specified for the provided parameters.

IsTrue(bool, string, string)

Checks for whether `condition == true`.

```
public bool IsTrue(bool condition, string problemReasonResolutionMessage,  
string argumentName)
```

Parameters

`condition` [bool](#)

Condition to be checked.

`problemReasonResolutionMessage` [string](#)

Validation failure message describing the `problem`, `reason`, and `resolution`.

`argumentName` [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

Returns

[bool](#)

[true](#) if the condition == [true](#); Otherwise, [false](#).

LessOrEqual<T>(T, T, string)

Checks if a numeric value is less or equal to a bound.

```
public bool LessOrEqual<T>(T value, T boundary, string argumentName) where T  
: IComparable<T>
```

Parameters

`value` T

The value to be checked against the provided boundary.

`boundary` T

The boundary that the value must be greater than or equal to.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

Returns

[bool](#)

[true](#) if the **value** is less than or equal to the provided boundary; otherwise, [false](#).

Type Parameters

T

The type specified for the provided parameters.

LessThan<T>(T, T, string)

Checks if a numeric value is less than the provided boundary.

```
public bool LessThan<T>(T value, T boundary, string argumentName) where T : IComparable<T>
```

Parameters

value T

The value to be checked against the provided boundary.

boundary T

The boundary that the value must be less than.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

Returns

[bool](#)

[true](#) if the **value** is less than the provided boundary; otherwise, [false](#).

Type Parameters

T

The type specified for the provided parameters.

MethodHandle<T>(string, string, out MethodHandle<T>)

Checks for valid method handle spec and creates the object.

```
public bool MethodHandle<T>(string fullyQualifiedNamespace, string argumentName, out  
MethodHandle<T> method) where T : Delegate
```

Parameters

fullyQualifiedNamespace [string](#)

Fully qualified name of the method to be checked.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

method [MethodHandle](#)<T>

Delegate to be created if the fullyQualifiedNamespace is found to be a valid method handle spec.

Returns

[bool](#)

[true](#) if the **fullyqualifiedNamespace** was found to match an existing method and the new delegate was created; otherwise, [false](#).

Type Parameters

T

The target delegate and it's accompanying parameter types.

MultiCondition<T>(string, Func<string, T>, string, out T[], int?)

Checks multi-condition validity and creates the data array.

```
public bool MultiCondition<T>(string csv, Func<string, T> parser, string argumentName, out  
T[] conditions, int? referenceCount = null)
```

Parameters

csv [string](#)

Comma separated values to split and parse.

parser [Func](#)<[string](#), T>

Delegate used to parse comma separated list.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

conditions [T\[\]](#)

Output Array of parsed values sourced from the provided string.

referenceCount [int](#)?

Optional. If specified, the reference count to verify. Will report an error if the resulting array size is >1 (SingleCondition) and does not match (MultiCondition).

Returns

[bool](#)

[true](#) if the output array was successfully created; otherwise, [false](#).

Type Parameters

T

The target (output) type of the parser function.

MultiCondition<TEnum>(string, string, out TEnum[], int?)

Checks multi-condition validity and creates the data array.

```
public bool MultiCondition<TEnum>(string csv, string argumentName, out TEnum[] conditions, int? referenceCount = null) where TEnum : struct, Enum
```

Parameters

csv [string](#)

Comma separated values to split and parse. Case-Insensitive, specify only the enum member part.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

conditions TEnum[]

Output Array of parsed values sourced from the provided string.

referenceCount [int](#)?

Optional. If specified, the reference count to verify. Will report an error if the resulting array size is >1 (SingleCondition) and does not match (MultiCondition).

Returns

[bool](#)

[true](#) if the output array was successfully created; otherwise, [false](#).

Type Parameters

TEnum

Pattern(Pattern, string, out PatternInfo, bool)

Checks for valid pattern spec and creates patternInfo object.

```
public bool Pattern(Pattern pattern, string argumentName, out PatternInfo patternInfo, bool threading = true)
```

Parameters

pattern Pattern

The pattern file to check.

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

patternInfo [PatternInfo](#)

A new [PatternInfo](#) object containing the provided pattern.

threading [bool](#)

Optional. Indicate whether threading should be used. Validation will fail if threading is not supported by the pattern.

Returns

[bool](#)

[true](#) if the **pattern** exists and creates the new [PatternInfo\(\)](#) object; otherwise, [false](#).

Pins(PinList, string, out Pins)

Checks for C#RA supported pin spec and creates the object.

```
public bool Pins(PinList pinList, string argumentName, out Pins pins)
```

Parameters

pinList PinList

argumentName [string](#)

The argument name used to indicate to IG-XL which test instance parameter failed.

pins [Pins](#)

A new [Pins](#) object containing all of the resolved pins.

Returns

bool ↴

true ↴ if the pinList was comprised of valid pins and creates the Pins() object; otherwise, false ↴.

Enum AlertOutputTarget

Namespace: [Csra](#)

Assembly: Csra.dll

The available output targets for Alert Service messages.

```
[Flags]
public enum AlertOutputTarget
```

Fields

Datalog = 2

File = 4

OutputWindow = 1

Class BiDictionary<TKey, TValue>

Namespace: [Csra](#)

Assembly: Csra.dll

Represents a bidirectional dictionary that allows lookups in both directions.

```
[Serializable]
public class BiDictionary<TKey, TValue> : IEnumerable<KeyValuePair<TKey,
TValue>>, IEnumerable
```

Type Parameters

TKey

The type of the keys.

TValue

The type of the values.

Inheritance

[object](#) ← BiDictionary<TKey, TValue>

Implements

[IEnumerable](#)<[KeyValuePair](#)<TKey, TValue>>, [IEnumerable](#)

Inherited Members

[object.ToString\(\)](#), [object.Equals\(object\)](#), [object.Equals\(object, object\)](#),
[object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#)

Properties

Count

Gets the number of key/value pairs contained in the BiDictionary.

```
public int Count { get; }
```

Property Value

[int ↗](#)

Methods

Add(TKey, TValue)

Adds a key/value pair to the BiDictionary if the key does not already exist.

```
public void Add(TKey key, TValue value)
```

Parameters

key TKey

The key to be added.

value TValue

The value of the element to add.

Clear()

Removes all keys and values from the BiDictionary.

```
public void Clear()
```

ContainsKey(TKey)

Determines whether the BiDictionary contains the specified key.

```
public bool ContainsKey(TKey key)
```

Parameters

key TKey

The key to locate in the BiDictionary.

Returns

[bool](#)

[true](#) if the BiDictionary contains an element with the specified key; otherwise, [false](#).

ContainsValue(TValue)

Determines whether the BiDictionary contains the specified value.

```
public bool ContainsValue(TValue value)
```

Parameters

value TValue

The value to locate in the BiDictionary.

Returns

[bool](#)

[true](#) if the BiDictionary contains an element with the specified value; otherwise, [false](#).

GetEnumerator()

Gets a collection containing the keys in the BiDictionary.

```
public IEnumarator<KeyValuePair<TKey, TValue>> GetEnumerator()
```

Returns

[IEnumarator](#)<[KeyValuePair](#)<TKey, TValue>>

RemoveByKey(TKey)

Removes the key/value pair with the specified key from the BiDictionary.

```
public bool RemoveByKey(TKey key)
```

Parameters

key TKey

The key of the element to remove.

Returns

[bool](#)

[true](#) if the element is successfully found and removed; otherwise [false](#).

RemoveByValue(TValue)

Removes the key/value pair with the specified value from the BiDictionary.

```
public bool RemoveByValue(TValue value)
```

Parameters

value TValue

Returns

[bool](#)

[true](#) if the element is successfully found and removed; otherwise [false](#).

TryGetKey(TValue, out TKey)

Gets the key associated with the specified value.

```
public bool TryGetKey(TValue value, out TKey key)
```

Parameters

value TValue

The value of the key to get.

key TKey

When this methods returns, contains the key associated with the specified value, if the value is found; otherwise, false.

Returns

bool

[true](#) if the BiDictionary contains an element with the specified value; otherwise, [false](#).

TryGetValue(TKey, out TValue)

Gets the value associated with the specified key.

```
public bool TryGetValue(TKey key, out TValue value)
```

Parameters

key TKey

The key of the value to get.

value TValue

When this methods returns, contains the value associated with the specified key, if the key is found; otherwise, false.

Returns

bool

[true](#) if the BiDictionary contains an element with the specified key; otherwise, [false](#).

Class CSRATransactionConfig

Namespace: [Csra](#)

Assembly: Csra.dll

```
public class CSRATransactionConfig : ITransactionConfig
```

Inheritance

[object](#) ← CSRATransactionConfig

Implements

[ITransactionConfig](#)

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Constructors

CSRATransactionConfig()

```
public CSRATransactionConfig()
```

Properties

DefaultPort

```
public string DefaultPort { get; set; }
```

Property Value

[string](#)

Valid

```
public bool Valid { get; }
```

Property Value

[bool](#)

Methods

AddPin(string, string, string, string, string)

```
public bool AddPin(string pin, string atePin = "", string type = "", string defaultState = "", string initState = "")
```

Parameters

pin [string](#)

atePin [string](#)

type [string](#)

defaultState [string](#)

initState [string](#)

Returns

[bool](#)

Class CSRATransactionConfig.Port

Namespace: [Csra](#)

Assembly: Csra.dll

```
public class CSRATransactionConfig.Port
```

Inheritance

[object](#) ← CSRATransactionConfig.Port

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Constructors

Port(string, string, List<string>, string)

```
public Port(string name, string protocol, List<string> pins, string registerMap)
```

Parameters

name [string](#)

protocol [string](#)

pins [List](#)<[string](#)>

registerMap [string](#)

Fields

Name

```
public string Name
```

Field Value

[string](#)

Pins

```
public List<string> Pins
```

Field Value

[List](#) <[string](#)>

Protocol

```
public string Protocol
```

Field Value

[string](#)

RegisterMap

```
public string RegisterMap
```

Field Value

[string](#)

Class CoreInstanceTestResult

Namespace: [Csra](#)

Assembly: Csra.dll

Class to store the test result of a core instance, part of [ScanNetworkPatternResults](#) of a ScanNetwork pattern(set).

```
[Serializable]
public class CoreInstanceTestResult : IEnumerable<string>, IEnumerable
```

Inheritance

[object](#) ← CoreInstanceTestResult

Implements

[IEnumerable](#)<[string](#)>, [IEnumerable](#)

Inherited Members

[object.ToString\(\)](#), [object.Equals\(object\)](#), [object.Equals\(object, object\)](#),
[object.ReferenceEquals\(object, object\)](#), [object.GetHashCode\(\)](#), [object.GetType\(\)](#),
[object.MemberwiseClone\(\)](#)

Constructors

CoreInstanceTestResult(string, IEnumerable<string>)

Creates a new [CoreInstanceTestResult](#) object for a core instance.

```
public CoreInstanceTestResult(string coreInstanceName, IEnumerable<string> iclInstanceNames)
```

Parameters

coreInstanceName [string](#)

The core instance for which this result is representing.

iclInstanceNames [IEnumerable](#)<[string](#)>

The names of icl instances that belongs to this core instance.

Properties

ErrorStatus

Returns the per-site error status if any. 0 = no error.

```
public Site<int> ErrorStatus { get; }
```

Property Value

Site<[int](#)>

InstanceName

Gets the core instance name.

```
public string InstanceName { get; }
```

Property Value

[string](#)

IsFailed

Gets or sets the per-site test result of the core instance. True = failed, False = passed.

```
public Site<bool> IsFailed { get; set; }
```

Property Value

Site<[bool](#)>

IsResultValid

Result is valid if core instance is fully tested. i.e. all icl instances under this core instance are tested.

```
public Site<bool> IsResultValid { get; set; }
```

Property Value

Site<[bool](#)>

TestName

Gets or sets the test name for datalogging the result of this core instance.

```
public string TestName { get; set; }
```

Property Value

[string](#)

TestNumber

Gets or sets the test number for datalogging the result of this core instance.

```
public int? TestNumber { get; set; }
```

Property Value

[int](#)?

Methods

GetEnumerator()

Returns an enumerator that iterates through the collection of strings.

```
public IEnumerator<string> GetEnumerator()
```

Returns

IEnumerator <string>

An enumerator that can be used to iterate through the collection.

Class DcParameters

Namespace: [Csra](#)

Assembly: Csra.dll

```
public class DcParameters
```

Inheritance

[object](#) ← DcParameters

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Properties

BleederResistor

```
public bool? BleederResistor { get; set; }
```

Property Value

[bool](#)?

ClampHiV

```
public double? ClampHiV { get; set; }
```

Property Value

[double](#)?

ClampLoV

```
public double? ClampLoV { get; set; }
```

Property Value

double?

ComplianceBoth

```
public double? ComplianceBoth { get; set; }
```

Property Value

double?

ComplianceNegative

```
public double? ComplianceNegative { get; set; }
```

Property Value

double?

CompliancePositive

```
public double? CompliancePositive { get; set; }
```

Property Value

double?

Current

```
public double? Current { get; set; }
```

Property Value

[double](#)?

CurrentRange

```
public double? CurrentRange { get; set; }
```

Property Value

[double](#)?

ForceBandwidth

```
public double? ForceBandwidth { get; set; }
```

Property Value

[double](#)?

Gate

```
public bool? Gate { get; set; }
```

Property Value

[bool](#)?

HardwareAverage

```
public double? HardwareAverage { get; set; }
```

Property Value

double?

HighAccuracy

```
public bool? HighAccuracy { get; set; }
```

Property Value

bool?

MeterBandwidth

```
public double? MeterBandwidth { get; set; }
```

Property Value

double?

MeterCurrentRange

```
public double? MeterCurrentRange { get; set; }
```

Property Value

double?

MeterMode

```
public Measure? MeterMode { get; set; }
```

Property Value

[Measure?](#)

MeterVoltageRange

```
public double? MeterVoltageRange { get; set; }
```

Property Value

[double?](#)

Mode

```
public TLibOutputMode? Mode { get; set; }
```

Property Value

[TLibOutputMode?](#)

SettlingTime

```
public double? SettlingTime { get; set; }
```

Property Value

[double?](#)

SinkFoldLimit

```
public double? SinkFoldLimit { get; set; }
```

Property Value

[double](#)?

SinkOverloadLimit

```
public double? SinkOverloadLimit { get; set; }
```

Property Value

[double](#)?

SourceFoldLimit

```
public double? SourceFoldLimit { get; set; }
```

Property Value

[double](#)?

SourceOverloadLimit

```
public double? SourceOverloadLimit { get; set; }
```

Property Value

[double](#)?

Voltage

```
public double? Voltage { get; set; }
```

Property Value

double?

VoltageAlt

```
public double? VoltageAlt { get; set; }
```

Property Value

double?

VoltageAltOutput

```
public bool? VoltageAltOutput { get; set; }
```

Property Value

bool?

VoltageRange

```
public double? VoltageRange { get; set; }
```

Property Value

double?

Class DigitalPinsLevelsParameters

Namespace: [Csra](#)

Assembly: Csra.dll

```
public class DigitalPinsLevelsParameters
```

Inheritance

[object](#) ← DigitalPinsLevelsParameters

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Properties

DifferentialLevelsType

```
public ChDiffPinLevel? DifferentialLevelsType { get; set; }
```

Property Value

ChDiffPinLevel?

DifferentialLevelsValue

```
public double? DifferentialLevelsValue { get; set; }
```

Property Value

[double](#)?

DifferentialLevelsValues

```
public double[] DifferentialLevelsValues { get; set; }
```

Property Value

[double](#)[]

DifferentialLevelsValuesType

```
public TLibDiffLvlValType[] DifferentialLevelsValuesType { get; set; }
```

Property Value

[TLibDiffLvlValType](#)[]

LevelsDriverMode

```
public tlDriverMode? LevelsDriverMode { get; set; }
```

Property Value

tlDriverMode?

LevelsType

```
public ChPinLevel? LevelsType { get; set; }
```

Property Value

ChPinLevel?

LevelsValue

```
public double? LevelsValue { get; set; }
```

Property Value

double?

LevelsValuePerSite

```
public SiteDouble LevelsValuePerSite { get; set; }
```

Property Value

SiteDouble

LevelsValues

```
public PinListData LevelsValues { get; set; }
```

Property Value

PinListData

Class DigitalPinsParameters

Namespace: [Csra](#)

Assembly: Csra.dll

```
public class DigitalPinsParameters
```

Inheritance

[object](#) ← DigitalPinsParameters

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Properties

AlarmBehavior

```
public tlAlarmBehavior? AlarmBehavior { get; set; }
```

Property Value

tlAlarmBehavior?

AlarmType

```
public tlHSDMAlarm? AlarmType { get; set; }
```

Property Value

tlHSDMAlarm?

CalibrationExcluded

```
public bool? CalibrationExcluded { get; set; }
```

Property Value

[bool](#)?

CalibrationHighAccuracy

```
public bool? CalibrationHighAccuracy { get; set; }
```

Property Value

[bool](#)?

DisableCompare

```
public bool? DisableCompare { get; set; }
```

Property Value

[bool](#)?

DisableDrive

```
public bool? DisableDrive { get; set; }
```

Property Value

[bool](#)?

InitState

```
public ChInitState? InitState { get; set; }
```

Property Value

ChInitState?

StartState

```
public ChStartState? StartState { get; set; }
```

Property Value

ChStartState?

Class DigitalPinsTimingParameters

Namespace: [Csra](#)

Assembly: Csra.dll

```
public class DigitalPinsTimingParameters
```

Inheritance

[object](#) ← DigitalPinsTimingParameters

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Properties

AutoStrobeEnabled

```
public AutoStrobeEnableSel? AutoStrobeEnabled { get; set; }
```

Property Value

AutoStrobeEnableSel?

AutoStrobeNumSteps

```
public int? AutoStrobeNumSteps { get; set; }
```

Property Value

[int](#)?

AutoStrobeSamplesPerStep

```
public int? AutoStrobeSamplesPerStep { get; set; }
```

Property Value

[int](#)?

AutoStrobeStartTime

```
public double? AutoStrobeStartTime { get; set; }
```

Property Value

[double](#)?

AutoStrobeStepTime

```
public double? AutoStrobeStepTime { get; set; }
```

Property Value

[double](#)?

FreeRunningClockEnabled

```
public bool? FreeRunningClockEnabled { get; set; }
```

Property Value

[bool](#)?

FreeRunningClockFrequency

```
public double? FreeRunningClockFrequency { get; set; }
```

Property Value

double?

FreqCtrEnable

```
public FreqCtrEnableSel? FreqCtrEnable { get; set; }
```

Property Value

FreqCtrEnableSel?

FreqCtrEventSlope

```
public FreqCtrEventSlopeSel? FreqCtrEventSlope { get; set; }
```

Property Value

FreqCtrEventSlopeSel?

FreqCtrEventSource

```
public FreqCtrEventSrcSel? FreqCtrEventSource { get; set; }
```

Property Value

FreqCtrEventSrcSel?

FreqCtrInterval

```
public double? FreqCtrInterval { get; set; }
```

Property Value

[double](#)?

TimingClockOffset

```
public double? TimingClockOffset { get; set; }
```

Property Value

[double](#)?

TimingClockPeriod

```
public double? TimingClockPeriod { get; set; }
```

Property Value

[double](#)?

TimingDisableAllEdges

```
public bool? TimingDisableAllEdges { get; set; }
```

Property Value

[bool](#)?

TimingEdgeEnabled

```
public bool? TimingEdgeEnabled { get; set; }
```

Property Value

bool?

TimingEdgeSet

```
public string TimingEdgeSet { get; set; }
```

Property Value

string?

TimingEdgeTime

```
public double? TimingEdgeTime { get; set; }
```

Property Value

double?

TimingEdgeVal

```
public chEdge? TimingEdgeVal { get; set; }
```

Property Value

chEdge?

TimingOffsetEnabled

```
public bool? TimingOffsetEnabled { get; set; }
```

Property Value

bool?

TimingOffsetSelectedPerSite

```
public SiteLong TimingOffsetSelectedPerSite { get; set; }
```

Property Value

SiteLong

TimingOffsetType

```
public tIOffsetType? TimingOffsetType { get; set; }
```

Property Value

tIOffsetType?

TimingOffsetValue

```
public double? TimingOffsetValue { get; set; }
```

Property Value

double?

TimingOffsetValuePerSiteIndex

```
public int? TimingOffsetValuePerSiteIndex { get; set; }
```

Property Value

[int](#)?

TimingOffsetValuePerSiteValue

```
public SiteDouble TimingOffsetValuePerSiteValue { get; set; }
```

Property Value

SiteDouble

TimingRefOffset

```
public double? TimingRefOffset { get; set; }
```

Property Value

[double](#)?

TimingSetup1xDiagnosticCapture

```
public string TimingSetup1xDiagnosticCapture { get; set; }
```

Property Value

[string](#)?

TimingSrcSyncDataDelay

```
public double? TimingSrcSyncDataDelay { get; set; }
```

Property Value

double?

Class ExtensionMethods

Namespace: [Csra](#)

Assembly: Csra.dll

```
public static class ExtensionMethods
```

Inheritance

[object](#) ← ExtensionMethods

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

SingleOrAt<T>(T[], int)

Returns the single element of a sequence, or the element at the specified index if the sequence contains more than one element.

```
public static T SingleOrAt<T>(this T[] values, int index)
```

Parameters

values T[]

The sequence to return an element from.

index [int](#)

The zero-based index of the element to retrieve if the sequence contains more than one element.

Returns

T

The single element of the sequence, or the element at the specified index if the sequence contains more than one element.

Type Parameters

T

The type of the elements of the source sequence.

Class IclInstanceInfo

Namespace: [Csra](#)

Assembly: Csra.dll

Class to store the attributes of an icl instance, part of [ScanNetworkPatternInfo](#) for a ScanNetwork pattern(set).

```
[Serializable]  
public class IclInstanceInfo
```

Inheritance

[object](#) ← IclInstanceInfo

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Constructors

IclInstanceInfo(string)

Creating a new [IclInstanceInfo](#) object from the combined instance name: "\${ssh-icl-instance}@{core-instance}".

```
public IclInstanceInfo(string iclAndCoreInstanceNames)
```

Parameters

iclAndCoreInstanceNames [string](#)

The hybrid instance name following this format: "{ssh_icl_instance}@{core_instance}".

Remarks

This hybrid instance name is retrieved from IGXL API:

TheHdw.Digital.ScanNetworks[ScanNetworkMapping].InstanceNames.

IclInstanceInfo(string, string, string, bool, int?, string, string, int?, string, string, int?, double?, string, string)

Creating a new [IclInstanceInfo](#) object which is a member of a collection in [ScanNetworkPatternInfo](#).

```
public IclInstanceInfo(string sshInstanceName, string sshIclInstanceName, string coreInstanceName, bool isOnChipCompare = false, int? tckRatio = null, string contribPin = null, string contribLabel = null, int? contribOffset = null, string stickyPin = null, string stickyLabel = null, int? stickyOffset = null, double? stickyCycle = null, string globalGroupID = null, string representativeSsh = null)
```

Parameters

sshInstanceName [string](#)?

The name of the ssh-instance, `representative_ssh` use this instance name as reference.

sshIclInstanceName [string](#)?

The name of the ssh-icl-instance, ssn mapping section use this name as reference.

coreInstanceName [string](#)?

The name of the core-instance that this ssh-icl-instance is associated with.

isOnChipCompare [bool](#)?

Optional. The attribute that indicates if OnChipCompare is enabled for this instance.

`true`: OnChipCompare = on; `false`: OnChipCompare = off;

tckRatio [int](#)?

Optional. The speed ratio between the Scan Cycles over the Jtag Cycles.

contribPin [string](#)?

Optional. Pin name for modifying the `disable_contribution_bit`.

contribLabel [string](#)?

Optional. Pattern label for locating the `disable_contribution_bit`.

contribOffset [int](#)?

Optional. Offset of the `disable_contribution_bit` relative to the `ContribLabel`.

stickyPin [string](#)

Optional. Pin name for retrieving the `sticky_bit` status.

stickyLabel [string](#)

Optional. Label for locating the `sticky_bit`.

stickyOffset [int](#)?

Optional. Offset of the `sticky_bit` relative to the `stickyLabel`.

stickyCycle [double](#)?

Optional. Absolute cycle of the `sticky_bit` in the `ssn_end_pattern`.

globalGroupID [string](#)

Optional. Capture Global Group ID for this ssh-icl-instance.

representativeSsh [string](#)

Optional. The name of the ssh-instance that represent this instance.

Remarks

Users typically do not need to create new icl instances by calling this constructor manually. Instead, please use the [ScanNetworkPatternInfo](#) to create and manage all icl instances.

Properties

ContribLabel

Gets the label for the disable-contribution-bit in the `ssn_setup` pattern. Only valid if `OnChipCompare` is true.

```
public string ContribLabel { get; }
```

Property Value

[string](#)

ContribOffset

Gets the offset of the disable-contribution-bit relative to the contribution label. Only valid if OnChipCompare is true.

```
public int? ContribOffset { get; }
```

Property Value

[int](#)?

ContribPin

Gets the PIN for the disable-contribution-bits in the ssn_setup pattern. Only valid if OnChipCompare is true.

```
public string ContribPin { get; }
```

Property Value

[string](#)

CoreInstanceName

Gets the name of the core instance that this icl instance belongs to.

```
public string CoreInstanceName { get; }
```

Property Value

[string](#)

GlobalGroupID

Gets the capture-global-group ID that this icl instance belongs to.

```
public string GlobalGroupID { get; }
```

Property Value

[string](#)

IclInstanceName

Gets the name of the icl instance.

```
public string IclInstanceName { get; }
```

Property Value

[string](#)

IsOnChipCompare

Whether OnChipCompare is enabled on this icl instance. [false](#) means this icl needs TesterCompare.

```
public bool IsOnChipCompare { get; }
```

Property Value

[bool](#)

ModifyVectorData

Substring of the disable-contribution-bit to be patched. Only used when OnChipCompare is true.

```
public Site<string> ModifyVectorData { get; }
```

Property Value

[Site<\[string\]\(#\)>](#)

SshInstanceName

Gets the name of the ssh instance.

```
public string SshInstanceName { get; }
```

Property Value

[string](#)

StickyCycle

Gets the absolute cycle (module cycle) of the sticky-bit in the ssn_end pattern. Only valid if OnChipCompare is true.

```
public double? StickyCycle { get; }
```

Property Value

[double](#)?

StickyPin

Gets the PIN for the sticky-bit in the ssn_end pattern. Only valid if OnChipCompare is true.

```
public string StickyPin { get; }
```

Property Value

[string](#)

Methods

SetDisableContributionBit(char)

Sets or clears the disable-contribution-bit for all sites.

```
public void SetDisableContributionBit(char value)
```

Parameters

value [char](#)

The new state of the disable-contribution-bit that applies to all sites.

SetDisableContributionBit(int, char)

Sets or clears the disable-contribution-bit for a specified site.

```
public void SetDisableContributionBit(int site, char value)
```

Parameters

site [int](#)

The identifier of the site for which the disable contribution bit is being modified.

value [char](#)

The new state of the disable-contribution-bit.

SetDisableContributionBit(Site<char>)

Sets or clears the disable-contribution-bit for each site.

```
public void SetDisableContributionBit(Site<char> value)
```

Parameters

value Site<[char](#)>

The new states of the disable-contribution-bit for each site.

Class IclInstanceTestResult

Namespace: [Csra](#)

Assembly: Csra.dll

Class to store the metadata and test result of an icl instance, part of [ScanNetworkPatternResults](#) of a ScanNetwork pattern(set).

```
[Serializable]  
public class IclInstanceTestResult
```

Inheritance

[object](#) ← IclInstanceTestResult

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Constructors

IclInstanceTestResult(IclInstanceStateInfo)

Create a new [IclInstanceTestResult](#) for an icl instance.

```
public IclInstanceTestResult(IclInstanceStateInfo iclInstance)
```

Parameters

iclInstance [IclInstanceStateInfo](#)

The icl instance for which this result is representing.

Properties

CoreInstanceName

Gets the core instance name that this icl instance belongs to.

```
public string CoreInstanceName { get; }
```

Property Value

[string](#)

ErrorStatus

Returns the per-site error status if any. 0 = no error.

```
public Site<int> ErrorStatus { get; }
```

Property Value

[Site<int>](#)

InstanceName

Gets the icl instance name.

```
public string InstanceName { get; }
```

Property Value

[string](#)

IsFailed

Gets or sets the per-site test result of the icl instance. True = failed, False = passed.

```
public Site<bool> IsFailed { get; set; }
```

Property Value

Site<[bool](#)>

IsOnChipCompare

Returns true if this icl instance is an on-chip compare instance.

```
public bool IsOnChipCompare { get; }
```

Property Value

[bool](#)

IsResultValid

Result is valid if the icl instance is tested. i.e. not masked or set to disable-contribution.

```
public Site<bool> IsResultValid { get; set; }
```

Property Value

Site<[bool](#)>

TestName

Gets or sets the test name for datalogging the result of this icl instance.

```
public string TestName { get; set; }
```

Property Value

[string](#)

TestNumber

Gets or sets the test number for datalogging the result of this icl instance.

```
public int? TestNumber { get; set; }
```

Property Value

[int](#)?

Enum InitMode

Namespace: [Csra](#)

Assembly: Csra.dll

Events in the lifetime of a Setup that will cause a reset of the hardware state.

```
[Flags]
public enum InitMode
```

Fields

Creation = 0

OnProgramStarted = 1

Enum InstrumentDomain

Namespace: [Csra](#)

Assembly: Csra.dll

Functional domain available by the instrument.

```
public enum InstrumentDomain
```

Fields

Dc = 0

Digital = 1

Utility = 2

Enum InstrumentFeature

Namespace: [Csra](#)

Assembly: Csra.dll

Logical features provided through the instrument driver.

```
public enum InstrumentFeature
```

Fields

Dcvi = 1

Dcvs = 2

Digital = 3

Ppmu = 0

Utility = 4

Enum InstrumentType

Namespace: [Csra](#)

Assembly: Csra.dll

Physical instrument types.

```
public enum InstrumentType
```

Fields

NC = 0

SupportBoard = 6

UP2200 = 1

UPHP = 2

UVI264 = 3

UVS256 = 5

UVS64 = 4

Enum Kelvin

Namespace: [Csra](#)

Assembly: Csra.dll

```
public enum Kelvin
```

Fields

Force = 1

Sense = 2

Enum Measure

Namespace: [Csra](#)

Assembly: Csra.dll

```
public enum Measure
```

Fields

Current = 1

Voltage = 0

Class MethodHandleTargetAttribute

Namespace: [Csra](#)

Assembly: Csra.dll

Indicates that the target method is intended to be referenced via a method handle, such as a delegate.

```
[AttributeUsage(AttributeTargets.Method, AllowMultiple = false, Inherited = false)]
public class MethodHandleTargetAttribute : Attribute, _Attribute
```

Inheritance

[object](#) ← [Attribute](#) ← MethodHandleTargetAttribute

Implements

[Attribute](#)

Inherited Members

[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#) ,
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#) ,
[Attribute.GetCustomAttributes\(MemberInfo\)](#) , [Attribute.GetCustomAttributes\(MemberInfo, bool\)](#) ,
[Attribute.IsDefined\(MemberInfo, Type\)](#) , [Attribute.IsDefined\(MemberInfo, Type, bool\)](#) ,
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#) ,
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#) ,
[Attribute.GetCustomAttributes\(ParameterInfo\)](#) , [Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#) ,
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#) ,
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#) , [Attribute.IsDefined\(ParameterInfo, Type\)](#) ,
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#) , [Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#) ,
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#) ,
[Attribute.GetCustomAttributes\(Module, Type\)](#) , [Attribute.GetCustomAttributes\(Module\)](#) ,
[Attribute.GetCustomAttributes\(Module, bool\)](#) , [Attribute.GetCustomAttributes\(Module, Type, bool\)](#) ,
[Attribute.IsDefined\(Module, Type\)](#) , [Attribute.IsDefined\(Module, Type, bool\)](#) ,
[Attribute.GetCustomAttribute\(Module, Type\)](#) , [Attribute.GetCustomAttribute\(Module, Type, bool\)](#) ,
[Attribute.GetCustomAttributes\(Assembly, Type\)](#) ,
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly\)](#) ,
[Attribute.GetCustomAttributes\(Assembly, bool\)](#) , [Attribute.IsDefined\(Assembly, Type\)](#) ,
[Attribute.IsDefined\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttribute\(Assembly, Type\)](#) ,
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#) , [Attribute.Equals\(object\)](#) ,
[Attribute.GetHashCode\(\)](#) , [Attribute.Match\(object\)](#) , [Attribute.IsDefaultAttribute\(\)](#) ,
[Attribute.TypeId](#) , [object.ToString\(\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Class MethodHandle<T>

Namespace: [Csra](#)

Assembly: Csra.dll

Provides a flexible and type-safe mechanism for dynamically resolving and invoking delegates based on the execution context (e.g., debug vs. production).

```
[Serializable]  
public class MethodHandle<T> where T : Delegate
```

Type Parameters

T

The delegate type. Must inherit from [Delegate](#).

Inheritance

[object](#) ← MethodHandle<T>

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Constructors

MethodHandle(string)

Initializes a new instance of the [MethodHandle<T>](#) class.

```
public MethodHandle(string fullyQualifiedNamespace)
```

Parameters

fullyQualifiedNamespace [string](#)

The fully qualified name of the delegate to be created. ("Namespace.Class.Method")

Properties

Execute

Gets the resolved delegate instance based on the current execution context.

```
public T Execute { get; }
```

Property Value

T

FullyQualifiedName

Gets the fully qualified name of the method to bind to the delegate.

```
public string FullyQualifiedName { get; }
```

Property Value

[string](#)

Class PatternInfo

Namespace: [Csra](#)

Assembly: Csra.dll

Class to store information about a pattern.

```
[Serializable]  
public class PatternInfo
```

Inheritance

[object](#) ← PatternInfo

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

PatternInfo(string, bool)

Construct a new [PatternInfo](#) object and load the specified pattern.

```
public PatternInfo(string pattern, bool threading)
```

Parameters

pattern [string](#)

Specifies the pattern to load. This can be a pattern module name, pattern file, pattern set name, or a combination of these items.

threading [bool](#)

Indicate whether threading should be used. Validation will fail if threading is not supported by the pattern.

PatternInfo(Pattern, bool)

Construct a new [PatternInfo](#) object and load the specified pattern.

```
public PatternInfo(Pattern pattern, bool threading)
```

Parameters

pattern Pattern

Specifies the pattern to load. This can be a pattern module name, pattern file, pattern set name, or a combination of these items.

threading [bool](#)

Indicate whether threading should be used. Validation will fail if threading is not supported by the pattern.

Fields

MaxFlags

```
public const int MaxFlags = 15
```

Field Value

[int](#)

Name

The name of the pattern setup in IGXL.

```
public readonly string Name
```

Field Value

[string](#)

Properties

ClearFlags

Clear flag value for the pattern, default is all flags (0).

```
public int ClearFlags { get; set; }
```

Property Value

[int](#)

SetFlags

Set flag value for the pattern, default is cpuA (1).

```
public int SetFlags { get; set; }
```

Property Value

[int](#)

ThreadingEnabled

Whether or not the pattern can and will use threading.

```
public bool ThreadingEnabled { get; set; }
```

Property Value

[bool](#)

TimeDomain

Use this property to return a string that lists the time domain names associated with the specified PatternSpecification. If a pattern is not specified, this syntax returns a comma-separated list of all domain

names.

```
public string TimeDomain { get; }
```

Property Value

[string](#) ↗

Methods

Equals(PatternInfo)

Determines whether the specified [PatternInfo](#) instance is equal to the current instance.

```
public bool Equals(PatternInfo other)
```

Parameters

other [PatternInfo](#)

The [PatternInfo](#) instance to compare with the current instance.

Returns

[bool](#) ↗

true if the specified [PatternInfo](#) has the same public properties; otherwise, **false**.

Equals(object)

Determines whether the specified object is equal to the current [PatternInfo](#).

```
public override bool Equals(object obj)
```

Parameters

obj [object](#) ↗

The object to compare with the current instance.

Returns

[bool](#)

true if the specified object is a [PatternInfo](#) and has the same public properties; otherwise, **false**.

GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

[int](#)

A hash code for the current [PatternInfo](#).

Operators

operator ==(PatternInfo, PatternInfo)

Determines whether two [PatternInfo](#) instances are equal.

```
public static bool operator ==(PatternInfo left, PatternInfo right)
```

Parameters

left [PatternInfo](#)

The first [PatternInfo](#) to compare.

right [PatternInfo](#)

The second [PatternInfo](#) to compare.

Returns

[bool](#)

`true` if both instances are equal or both are `null`; otherwise, `false`.

operator !=(PatternInfo, PatternInfo)

Determines whether two [PatternInfo](#) instances are not equal.

```
public static bool operator !=(PatternInfo left, PatternInfo right)
```

Parameters

`left` [PatternInfo](#)

The first [PatternInfo](#) to compare.

`right` [PatternInfo](#)

The second [PatternInfo](#) to compare.

Returns

[bool](#) ↗

`true` if the instances are not equal; otherwise, `false`.

Class Pins

Namespace: [Csra](#)

Assembly: Csra.dll

Pins class - a collection of Pin objects.

```
[Serializable]
public class Pins : IEnumerable<Pins.Pin>, IEnumerable, IEquatable<Pins>
```

Inheritance

[object](#) ← Pins

Implements

[IEnumerable](#) <[Pins.Pin](#)>, [IEnumerable](#), [IEquatable](#) <[Pins](#)>

Inherited Members

[object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Constructors

Pins(string)

Construct a new [Pins](#) object. Resolves (nested) pin groups and lists.

```
public Pins(string pinList)
```

Parameters

pinList [string](#)

The pin list to create the [Pins](#) object for.

Methods

Add(string)

Add pins to the [Pins](#) object. Resolves (nested) pin groups and lists.

```
public void Add(string pinList)
```

Parameters

pinList [string](#)

The pin list to add to the [Pins](#) object.

ArrangePinSite<T>(IEnumerable<PinSite<T>>)

Combines multiple Teradyne.lgxl.Interfaces.Public.PinSite<T> objects into a single one maintaining this [Pins](#) object's pin sequence. Excessive objects are quietly ignored, a runtime exception is thrown for missing ones.

```
public PinSite<T> ArrangePinSite<T>(IEnumerable<PinSite<T>> pinSite)
```

Parameters

pinSite [IEnumerable](#)<PinSite<T>>

A collection of Teradyne.lgxl.Interfaces.Public.PinSite<T> instances to be combined and arranged.

Returns

PinSite<T>

A new Teradyne.lgxl.Interfaces.Public.PinSite<T> containing elements from all input Teradyne.lgxl.Interfaces.Public.PinSite<T> instances, arranged in the right order.

Type Parameters

T

The type of Teradyne.lgxl.Interfaces.Public.PinSite<T> elements.

ContainsDomain(InstrumentDomain)

Test if at least one pin has the specified domain.

```
public bool ContainsDomain(InstrumentDomain domain)
```

Parameters

[domain](#) [InstrumentDomain](#)

The instrument domain to look for.

Returns

[bool](#)

True if one or more pins have the domain.

ContainsDomain([InstrumentDomain](#), [out string](#))

Combined test and extraction of pins by instrument domain.

```
public bool ContainsDomain(InstrumentDomain domain, out string pinList)
```

Parameters

[domain](#) [InstrumentDomain](#)

The instrument domain to look for.

[pinList](#) [string](#)

A new [Pins](#) object only containing pins of the specified type.

Returns

[bool](#)

True if one or more pins have the domain.

ContainsFeature([InstrumentFeature](#))

Test if at least one pin has the specified feature.

```
public bool ContainsFeature(InstrumentFeature feature)
```

Parameters

feature [InstrumentFeature](#)

The instrument feature to look for.

Returns

[bool](#)

True if one or more pins have the feature.

ContainsFeature([InstrumentFeature](#), [out string](#))

Combined test and extraction of pins by instrument feature.

```
public bool ContainsFeature(InstrumentFeature feature, out string pinList)
```

Parameters

feature [InstrumentFeature](#)

The instrument feature to look for.

pinList [string](#)

A new [Pins](#) object only containing pins of the specified type.

Returns

[bool](#)

True if one or more pins have the feature.

ContainsType([InstrumentType](#))

Test if at least one pin is of the specified instrument type.

```
public bool ContainsType(InstrumentType type)
```

Parameters

type [InstrumentType](#)

The instrument type to look for.

Returns

[bool](#)

True if one or more pins have the type.

ContainsType(InstrumentType, out string)

Combined test and extraction of pins by instrument type.

```
public bool ContainsType(InstrumentType instrument, out string pinList)
```

Parameters

instrument [InstrumentType](#)

The instrument type to look for.

pinList [string](#)

A new [Pins](#) object only containing pins of the specified instrument type.

Returns

[bool](#)

True if one or more pins have the type.

Count()

Return the number of pins in the [Pins](#) object.

```
public int Count()
```

Returns

[int](#)

Equals(Pins)

Determines whether the specified [Pins](#) instance is equal to the current instance.

```
public bool Equals(Pins other)
```

Parameters

[other](#) [Pins](#)

The [Pins](#) instance to compare with the current instance.

Returns

[bool](#)

[true](#) if the specified [Pins](#) contains the same pins in the same order; otherwise, [false](#).

Equals(object)

Determines whether the specified object is equal to the current [Pins](#).

```
public override bool Equals(object obj)
```

Parameters

[obj](#) [object](#)

The object to compare with the current instance.

Returns

[bool](#)

`true` if the specified object is a [Pins](#) and contains the same pins in the same order; otherwise, `false`.

ExtractByDomain([InstrumentDomain](#))

Extract pins by instrument domain.

```
public Pins ExtractByDomain(InstrumentDomain domain)
```

Parameters

`domain` [InstrumentDomain](#)

The instrument domain to look for.

Returns

[Pins](#)

A new [Pins](#) object only containing pins with the specified domain.

ExtractByFeature([InstrumentFeature](#))

Extract pins by instrument feature.

```
public Pins ExtractByFeature(InstrumentFeature feature)
```

Parameters

`feature` [InstrumentFeature](#)

The instrument feature to look for.

Returns

[Pins](#)

A new [Pins](#) object only containing pins with the specified feature.

ExtractByType([InstrumentType](#))

Extract pins by instrument type.

```
public Pins ExtractByType(InstrumentType type)
```

Parameters

type [InstrumentType](#)

The instrument type to look for.

Returns

[Pins](#)

A new [Pins](#) object only containing pins of the specified instrument type.

ExtractRange([int](#), [int](#))

Extract a range of pins from the [Pins](#) object.

```
public Pins ExtractRange(int start, int count)
```

Parameters

start [int](#)

The index of the first pin to extract.

count [int](#)

The total number of pins to extract.

Returns

[Pins](#)

A new [Pins](#) object only containing the subset of pins.

GetEnumerator()

Get an enumerator for the [Pins](#) object to support

`foreach`

```
public IEnumrator<Pins.Pin> GetEnumerator()
```

Returns

[IEnumrator](#) <[Pins.Pin](#)>

GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

[int](#)

A hash code for the current [Pins](#).

Join(Pins[])

Combines a collection of [Pins](#) objects into a single [Pins](#) object.

```
public static Pins Join(Pins[] pinGroups)
```

Parameters

`pinGroups` [Pins](#)[]

An array of [Pins](#) objects.

Returns

[Pins](#)

A new [Pins](#) object with all pins combined.

Sort()

Rearranges the pins in the [Pins](#) object by name (in place).

```
public void Sort()
```

Sort<TKey>(Func<Pin, TKey>)

Rearranges the pins in the [Pins](#) object by the specified key (in place).

```
public void Sort<TKey>(Func<Pins.Pin, TKey> selector)
```

Parameters

selector [Func](#)<[Pins.Pin](#), TKey>

The selector delegate creating the sort key.

Type Parameters

TKey

The target key type for the selector delegate.

ToString()

Convert the [Pins](#) object to a comma-separated list of all pin names.

```
public override string ToString()
```

Returns

[string](#) ↗

A comma-separated list of all pin names.

Operators

operator ==(Pins, Pins)

Determines whether two [Pins](#) instances are equal.

```
public static bool operator ==(Pins left, Pins right)
```

Parameters

left [Pins](#)

The first [Pins](#) to compare.

right [Pins](#)

The second [Pins](#) to compare.

Returns

[bool](#) ↗

true if both instances are equal or both are **null**; otherwise, **false**.

operator !=(Pins, Pins)

Determines whether two [Pins](#) instances are not equal.

```
public static bool operator !=(Pins left, Pins right)
```

Parameters

left [Pins](#)

The first [Pins](#) to compare.

right Pins

The second [Pins](#) to compare.

Returns

[bool](#) ↗

true if the instances are not equal; otherwise, **false**.

Class Pins.Pin

Namespace: [Csra](#)

Assembly: Csra.dll

Pin class - individual hardware pins with features.

```
[Serializable]
public class Pins.Pin : IEquatable<Pins.Pin>
```

Inheritance

[object](#) ← Pins.Pin

Implements

[IEquatable](#)<[Pins.Pin](#)>

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object, object\)](#) , [object.ReferenceEquals\(object, object\)](#) ,
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#)

Constructors

Pin(string)

Construct a new [Pins.Pin](#) object.

```
public Pin(string name)
```

Parameters

name [string](#)

The pin name.

Properties

Domains

A collection of functional domains this pin supports.

```
public List<InstrumentDomain> Domains { get; }
```

Property Value

[List](#) <[InstrumentDomain](#)>

Features

A collection of features this pin supports.

```
public List<InstrumentFeature> Features { get; }
```

Property Value

[List](#) <[InstrumentFeature](#)>

Name

The (resolved) pin name.

```
public string Name { get; }
```

Property Value

[string](#)

Type

The instrument type of the pin.

```
public InstrumentType Type { get; }
```

Property Value

Methods

Equals(Pin)

Determines whether the specified [Pins.Pin](#) instance is equal to the current instance.

```
public bool Equals(Pins.Pin other)
```

Parameters

other [Pins.Pin](#)

The [Pins.Pin](#) instance to compare with the current instance.

Returns

[bool](#) ↗

true if the specified [Pins.Pin](#) has the same [Name](#); otherwise, **false**.

Equals(object)

Determines whether the specified object is equal to the current [Pins.Pin](#).

```
public override bool Equals(object obj)
```

Parameters

obj [object](#) ↗

The object to compare with the current instance.

Returns

[bool](#) ↗

true if the specified object is a [Pins.Pin](#) and has the same [Name](#); otherwise, **false**.

GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

[int](#)

A hash code for the current [Pins.Pin](#).

GetInstrumentName([InstrumentType](#))

This method is public but intended for internal use only.

```
public static string GetInstrumentName(InstrumentType type)
```

Parameters

[type](#) [InstrumentType](#)

The Type to convert.

Returns

[string](#)

The corresponding type string.

GetType([string](#))

This method is public but intended for internal use only.

```
public static InstrumentType GetType(string typeString)
```

Parameters

`typeString` [string](#)

The type string to convert.

Returns

[InstrumentType](#)

The corresponding Type.

Operators

`operator ==(Pin, Pin)`

Determines whether two [Pins.Pin](#) instances are equal.

```
public static bool operator ==(Pins.Pin left, Pins.Pin right)
```

Parameters

`left` [Pins.Pin](#)

The first [Pins.Pin](#) to compare.

`right` [Pins.Pin](#)

The second [Pins.Pin](#) to compare.

Returns

[bool](#)

`true` if both instances are equal or both are `null`; otherwise, `false`.

`operator !=(Pin, Pin)`

Determines whether two [Pins.Pin](#) instances are not equal.

```
public static bool operator !=(Pins.Pin left, Pins.Pin right)
```

Parameters

left [Pins.Pin](#)

The first [Pins.Pin](#) to compare.

right [Pins.Pin](#)

The second [Pins.Pin](#) to compare.

Returns

[bool](#) ↗

true if the instances are not equal; otherwise, **false**.

Enum ScanNetworkDatalogOption

Namespace: [Csra](#)

Assembly: Csra.dll

Specifies the options for logging ScanNetwork test results from object [ScanNetworkPatternResults](#).

```
[Flags]
public enum ScanNetworkDatalogOption
```

Fields

LogByCoreInstance = 2

LogByIclInstance = 4

LogFailedInstancesOnly = 8

LogPatGenWithFTR = 1

LogVerboseInfoByDTR = 16

Remarks

This enumeration provides different logging strategies that can be used to capture ScanNetwork test results. Each option represents a preference of logging, and can be combined to allow for flexibility in how data is recorded and analyzed.

Class ScanNetworkPatternInfo

Namespace: [Csra](#)

Assembly: Csra.dll

Class to store ScanNetwork information about a ScanNetwork pattern(set).

```
[Serializable]
public class ScanNetworkPatternInfo
```

Inheritance

[object](#) ← ScanNetworkPatternInfo

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Constructors

ScanNetworkPatternInfo(Pattern)

Construct a new [ScanNetworkPatternInfo](#) object and load the specified pattern(set).

```
public ScanNetworkPatternInfo(Pattern patternSetName)
```

Parameters

patternSetName Pattern

Specifies the ScanNetwork pattern(set) to load. This can be the name of a pattern file or a pattern set.

ScanNetworkPatternInfo(Pattern, string)

Construct a new [ScanNetworkPatternInfo](#) object and load the specified pattern(set).

```
public ScanNetworkPatternInfo(Pattern patternSetName, string concatenatedPatternCsvFileName)
```

Parameters

patternSetName Pattern

Specifies the ScanNetwork pattern(set) to load. This can be the name of a pattern file or a pattern set.

concatenatedPatternCsvFileName [string](#)

Specifies the Csv file that comes with the concatenated(set+payload+end) ScanNetwork pattern.

ScanNetworkPatternInfo(Pattern, string, string)

Construct a new [ScanNetworkPatternInfo](#) object and load the specified pattern(set).

```
public ScanNetworkPatternInfo(Pattern patternSetName, string setupPatternCsvFileName,  
string endPatternCsvFileName)
```

Parameters

patternSetName Pattern

Specifies the ScanNetwork pattern(set) to load. This can be the name of a pattern file or a pattern set.

setupPatternCsvFileName [string](#)

Specifies the Csv file that comes with the ScanNetwork_setup pattern.

endPatternCsvFileName [string](#)

Specifies the Csv file that comes with the ScanNetwork_end pattern

Properties

CaptureGlobalGroup

[[readonly](#)] The dictionary that contains the pattern's capture-global-group mapping, which associates each global group ID with a list of icl instances.

The key of the dictionary is the ID of the capture-global-group and the value is the [List<T>](#) of icl instances that belong to the capture-global-group.

```
public Dictionary<string, List<string>> CaptureGlobalGroup { get; }
```

Property Value

[Dictionary](#)<string, List<string>>

CoreInstance

[[readonly](#)] The dictionary that contains the names of all the cores that are tested in this pattern, and the names of icl instances under each core instance.

The key of the dictionary is the name of the core instance and the value is the [List<T>](#) of icl instances that belong to the core.

```
public Dictionary<string, List<string>> CoreInstance { get; }
```

Property Value

[Dictionary](#)<string, List<string>>

IclInstance

[[readonly](#)] The dictionary that contains the information of every ICL instance that are tested in this pattern.

The key is the name of the ICL instance and the value is the associated [SshIclInstanceInfo](#) object that contains all the configuration and attributes of that ICL instance.

```
public Dictionary<string, IclInstanceState> IclInstance { get; }
```

Property Value

[Dictionary](#)<string, [IclInstanceState](#)>

PatternSetName

[[readonly](#)] The name of the pattern(set) in IGXL.

```
public string PatternSetName { get; }
```

Property Value

[string](#)

ScanNetworkMapping

[[readonly](#)] The name of the ScanNetwork map file(csv). This name is used for masking certain time slots on scan out pins by applying TesterCompare Masks. TesterCompare related methods are all under: [TheHdw.Digital.ScanNetwork\[ScanNetworkMapping\]](#)

```
public string ScanNetworkMapping { get; }
```

Property Value

[string](#)

SetupPatternName

[[readonly](#)] The name of the pattern file that contains the contribution bits. This pattern will be modified during OCComp Diagnosis reburst.

```
public string SetupPatternName { get; }
```

Property Value

[string](#)

Methods

ClearAllDisableContributionBits()

Clears all disable_contribution_bits, resetting the state to its default.

```
public void ClearAllDisableContributionBits()
```

Remarks

This method affects the internal state by removing any flags that disable contributions. It should be used when a full reset of contribution settings is required.

GetScanNetworkPatternResults()

Returns a [ScanNetworkPatternResults](#) object that contains the test results of the latest execution of this pattern.

```
public ScanNetworkPatternResults GetScanNetworkPatternResults()
```

Returns

[ScanNetworkPatternResults](#)

A [ScanNetworkPatternResults](#) object that contains the test results of the latest execution of this pattern.

SetAllDisableContributionBits()

Sets all disable_contribution_bits in the ScanNetwork_setup pattern, disabling all OnChipCompare icl-instances from contributing to the ScanNetwork bus output.

```
public void SetAllDisableContributionBits()
```

Class ScanNetworkPatternResults

Namespace: [Csra](#)

Assembly: Csra.dll

Class to store the per core/icl instance test results of a ScanNetwork pattern(set).

```
[Serializable]
public class ScanNetworkPatternResults
```

Inheritance

[object](#) ← ScanNetworkPatternResults

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Constructors

ScanNetworkPatternResults(ScanNetworkPatternInfo)

Create a new instance of [ScanNetworkPatternResults](#) that can hold the per icl instance test result from a ScanNetwork pattern.

```
public ScanNetworkPatternResults(ScanNetworkPatternInfo scanNetworkPatternInfo)
```

Parameters

scanNetworkPatternInfo [ScanNetworkPatternInfo](#)

The [ScanNetworkPatternInfo](#) object for the ScanNetwork pattern

Properties

CoreInstance

Gets the dictionary of test results for each core instances, indexed by instance name.

```
public Dictionary<string, CoreInstanceTestResult> CoreInstance { get; }
```

Property Value

[Dictionary](#)<string, CoreInstanceTestResult>

IclInstance

Gets the dictionary of test results for each icl-instance, indexed by instance name.

```
public Dictionary<string, IclInstanceTestResult> IclInstance { get; }
```

Property Value

[Dictionary](#)<string, IclInstanceTestResult>

TestConditions

User-Defined Test Conditions, such as Voltage/Frequency/Retest#, etc.

```
public string TestConditions { get; }
```

Property Value

[string](#)

Methods

GetFailedCoreInstanceList()

Gets the list of failed core instances for each site.

```
public Site<List<string>> GetFailedCoreInstanceList()
```

Returns

Site<[List](#)<[string](#)>>

Per site value is a list of failed core instance names.

GetFailedCoreInstanceList(int)

Gets the list of failed core instances for a specified site.

```
public List<string> GetFailedCoreInstanceList(int site)
```

Parameters

site [int](#)

The site number.

Returns

[List](#)<[string](#)>

A list of failed core instance names for the specified site.

Class Setup

Namespace: [Csra](#)

Assembly: Csra.dll

Setup - a named collection of settings.

```
[Serializable]  
public class Setup
```

Inheritance

[object](#) ← Setup

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Constructors

Setup(string)

Creates a new setup with the specified name.

```
public Setup(string name)
```

Parameters

name [string](#)

The setup name - an arbitrary, case sensitive string that can include numbers, spaces and special characters. Empty and null strings are not supported.

Properties

Count

Gets the number of settings contained in the setup.

```
public int Count { get; }
```

Property Value

[int](#)

Name

Gets the name of the setup.

```
public string Name { get; }
```

Property Value

[string](#)

Methods

Add(ISetting)

Adds a setting to the setup.

```
public void Add(ISetting setting)
```

Parameters

setting [ISetting](#)

The setting to add.

Class SsnCsvFile

Namespace: [Csra](#)

Assembly: Csra.dll

Class to parse information from an ssn.csv file. Only for use in constructor method of [ScanNetworkPatternInfo](#).

```
[Serializable]  
public class SsnCsvFile
```

Inheritance

[object](#) ← SsnCsvFile

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Constructors

SsnCsvFile(string, string)

Construct a new [SsnCsvFile](#) object and load the specified _ssn.csv file.

```
public SsnCsvFile(string fileName, string instanceKey = "Ssh instance")
```

Parameters

fileName [string](#)

The _ssn.csv file name, which is generated by ATEGEN along with the atp file.

instanceKey [string](#)

Optional. The **Key** attribute for indexing the instances. Must be an attribute whose value is unique across all instances. By default it's the 3rd column, which is "Ssh instance".

Properties

ContribLabel

The label is used for modifying the disable-contribution-bits in the ssn_setup pattern.

```
public string ContribLabel { get; }
```

Property Value

[string](#)

ContribPin

The pin list is used for modifying the disable-contribution-bits in the ssn_setup pattern.

```
public string ContribPin { get; }
```

Property Value

[string](#)

CsvVersion

The version of the ssn setup csv file. It shall appear in the first line of the csv file, which starts with "//SSN instances". Default to "v1.0" if not specified.

```
public string CsvVersion { get; }
```

Property Value

[string](#)

InstanceKey

The attribute name used as the key for indexing instances.

```
public string InstanceKey { get; }
```

Property Value

[string](#)

SshInstances

Dictionary of SSH instances, where the default Attribute for key is the icl-instance name and the value is a dictionary of attribute names and values.

```
public Dictionary<string, Dictionary<string, string>> SshInstances { get; }
```

Property Value

[Dictionary](#)<[string](#), [Dictionary](#)<[string](#), [string](#)>>

StickyPin

The pin list that output the sticky bits in the ssn end pattern.

```
public string StickyPin { get; }
```

Property Value

[string](#)

TckRatio

The TCK ratio of the SSN pattern. It is used to determine the number of vectors to be modified for each disable-contribution-bit in the ssn_setup pattern. In v1.0, the attribute name was "Num bits", in v2025.7 it is changed to "Tck ratio".

```
public string TckRatio { get; }
```

Property Value

[string](#) ↗

Enum TLibDiffLvlValType

Namespace: [Csra](#)

Assembly: Csra.dll

```
public enum TLibDiffLvlValType
```

Fields

IOH = 13

IOL = 12

VCH = 10

VCL = 9

VICM = 3

VID = 0

VOD = 6

VT = 11

VocmTyp = 15

VodTyp = 14

dVICM0 = 4

dVICM1 = 5

dVID0 = 1

dVID1 = 2

dVOD0 = 7

dVOD1 = 8

Enum TLibOutputMode

Namespace: [Csra](#)

Assembly: Csra.dll

```
public enum TLibOutputMode
```

Fields

ForceCurrent = 1

ForceVoltage = 0

HighImpedance = 2

Enum TransactionType

Namespace: [Csra](#)

Assembly: Csra.dll

Transaction type to use with the TransactionService.

```
public enum TransactionType
```

Fields

CsraGeneric = 1

PortBridge = 0

Namespace Demo_CSRA

Namespaces

[Demo_CSRA.Continuity](#)

[Demo_CSRA.Functional](#)

[Demo_CSRA.Leakage](#)

[Demo_CSRA.Parametric](#)

[Demo_CSRA.Resistance](#)

[Demo_CSRA.ScanNetwork](#)

[Demo_CSRA.Search](#)

[Demo_CSRA.SupplyCurrent](#)

[Demo_CSRA.Timing](#)

[Demo_CSRA.Trim](#)

Classes

[CustomerExtensions](#)

[ExeInterposeClass](#)

This class contains empty Exec Interpose functions.

[SetupLoadClass](#)

[Showcase](#)

Namespace Demo_CSRA.Continuity

Classes

[Parametric](#)

[Supply](#)

Class Parametric

Namespace: [Demo_CSRA.Continuity](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class Parametric : TestCodeBase
```

Overview

TestClass for all digital continuity related TestMethods.

Platform Specifics

Uses per test-instance test class object persistence ([\[TestClass\(Creation.TestInstance\)\]](#) attribute).

Inheritance

[object](#) ← TestCodeBase ← Parametric

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
[TestCodeBase.TheHdw](#) , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
[TestCodeBase.ShouldRunPreBody](#) , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Parallel(PinList, double, double, double, double, string)

Checks if the tester resources have electrical contact with DUT and if any pin is short-circuited with another signal pin or power supply. The measurement is done parallel, all at once.

```
[TestMethod]  
[Steppable]  
[CustomValidation]  
public void Parallel(PinList pinList, double current, double clampVoltage, double  
voltageRange, double waitTime, string setup = "")
```

Parameters

pinList PinList

List of pin or pin group names.

current [double](#)

The current to force.

clampVoltage [double](#)

The value to clamp for force pin.

voltageRange [double](#)

The voltage range for measurement.

waitTime [double](#)

The wait time after forcing.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified **config**. For all pins specified in the **pinList** it disconnects any pin electronics, connects the dc path and turns on the gate.

The **Body** section applies a force `current` condition on all pins and performs a voltage measurement on all pins in parallel after the specified `waitTime`.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- digital pins in `pinList` have pin electronics connected
- any dc paths from pins in `pinList` are disconnected

Limitations

- support for non-uniform (mixed) instrument types in `pinList` not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Parallel(PinList pinList, double current, double clampVoltage, double
voltageRange, double waitTime, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(pinList, nameof(pinList), out _pins);
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        _containsDigitalPins = _pins.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
        TheLib.Setup.Dc.Connect(_pins);
    }

    if (ShouldRunBody) {
        TheLib.Setup.Dc.SetForceAndMeter(_pins, TLibOutputMode.ForceCurrent, current,
        current, clampVoltage, Measure.Voltage, voltageRange);
        TheLib.Execute.Wait(waitTime);
        _meas = TheLib.Acquire.Dc.Measure(_pins);
    }
}
```

```

    if (ShouldRunPostBody) {
        TheLib.Setup.Dc.Disconnect(_pins);
        if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
        TheLib.Datalog.TestParametric(_meas, current, "A");
    }
}

```

Serial(PinList, double, double, double, double, string)

Checks if the tester resources have electrical contact with DUT and if any pin is short-circuited with another signal pin or power supply. The measurement is done serially, one at a time.

```

[TestMethod]
[Steppable]
[CustomValidation]
public void Serial(PinList pinList, double current, double clampVoltage, double
voltageRange, double waitTime, string setup = "")

```

Parameters

pinList PinList

List of pin or pin group names.

current [double](#)

The current to force.

clampVoltage [double](#)

The value to clamp for force pin.

voltageRange [double](#)

The voltage range for measurement.

waitTime [double](#)

The wait time after forcing.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `config`. For all pins specified in the `pinList` it disconnects any pin electronics, connects the dc path.

The **Body** section applies a force 0V condition on all pins, then sequentially—for each pin—applies a force `current` condition, performs a voltage measurement after the specified `waitTime`, and resets the pin to force 0V.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

UltraFLEXplus, UltraFLEX

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- digital pins in `pinList` have pin electronics connected
- any dc paths from pins in `pinList` are disconnected

Implementation

```
[TestMethod, Steppable, CustomValidation]
public void Serial(PinList pinList, double current, double clampVoltage, double
voltageRange, double waitTime, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(pinList, nameof(pinList), out _pins);
        _pinSteps = _pins.Select(pin => new Pins(pin.Name)).ToArray();
        _containsDigitalPins = _pins.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
        TheLib.Setup.Dc.Connect(_pins);
    }
}
```

```

if (ShouldRunBody) {
    _meas = new();
    TheLib.Setup.Dc.ForceV(_pins, 0, outputModeVoltage: true);
    foreach (var pin in _pinSteps) {
        TheLib.Setup.Dc.SetForceAndMeter(pin, TLibOutputMode.ForceCurrent, current,
current, clampVoltage, Measure.Voltage, voltageRange, false);
        TheLib.Execute.Wait(waitTime);
        _meas.Add(TheLib.Acquire.Dc.Measure(pin).First());
        TheLib.Setup.Dc.ForceV(pin, 0, outputModeVoltage: true, gateOn: false);
    }
}

if (ShouldRunPostBody) {
    TheLib.Setup.Dc.Disconnect(_pins);
    if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
    TheLib.Datalog.TestParametric(_meas, current, "A");
}
}

```

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Class Supply

Namespace: [Demo_CSRA.Continuity](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class Supply : TestCodeBase
```

Overview

TestClass for all power supply continuity related TestMethods.

Platform Specifics

Uses per test-instance test class object persistence ([\[TestClass\(Creation.TestInstance\)\]](#) attribute).

Inheritance

[object](#) ← TestCodeBase ← Supply

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
[TestCodeBase.TheHdw](#) , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
[TestCodeBase.ShouldRunPreBody](#) , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Baseline(PinList, double, double, double, string)

Checks if the tester has electrical contact with pins of a DUT.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Baseline(PinList pinList, double forceVoltage, double currentRange, double
waitTime, string setup = "")
```

Parameters

pinList PinList

List of pin or pin group names.

forceVoltage [double ↗](#)

The force voltage value.

currentRange [double ↗](#)

The current range for measurement.

waitTime [double ↗](#)

The wait time after forcing.

setup [string ↗](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `config`. For all pins specified in the `pinList` it disconnects any pin electronics, connects the dc path and turns on the gate.

The **Body** section applies a force `forceVoltage` condition on all pins and performs a current measurement on all pins in parallel after the specified `waitTime`.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- any dc paths from pins in `pinList` are disconnected

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(PinList pinList, double forceVoltage, double currentRange, double
waitTime, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(pinList, nameof(pinList), out _pins);
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        _containsDigitalPins = _pins.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
        TheLib.Setup.Dc.Connect(_pins);
    }

    if (ShouldRunBody) {
        TheLib.Setup.Dc.SetForceAndMeter(_pins, TLibOutputMode.ForceVoltage, forceVoltage,
forceVoltage, currentRange, Measure.Current, currentRange);
        TheLib.Execute.Wait(waitTime);
        _meas = TheLib.Acquire.Dc.Measure(_pins);
    }

    if (ShouldRunPostBody) {
        TheLib.Setup.Dc.Disconnect(_pins);
        if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
        TheLib.Datalog.TestParametric(_meas, forceVoltage, "V");
    }
}
```


Namespace Demo_CSRA.Functional Classes

[Read](#)

[StaticPattern](#)

Class Read

Namespace: [Demo_CSRA.Functional](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class Read : TestCodeBase
```

Inheritance

[object](#) ← TestCodeBase ← Read

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
TestCodeBase.TheHdw , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
TestCodeBase.ShouldRunPreBody , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Baseline(Pattern, PinList, int, int, int, bool, bool, bool, string)

Executes a functional read from the device and logs the results.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Baseline(Pattern pattern, PinList readPins, int startIndex, int bitLength, int
wordLength, bool msbFirst, bool testFunctional, bool testValues, string setup = "")
```

Parameters

pattern Pattern

The pattern to be executed during the test.

readPins PinList

Pins for data read, must contain at least 1 digital pin.

startIndex [int](#)

Index to start read.

bitLength [int](#)

Length of data read.

wordLength [int](#)

Length of each data word from 1 to 32.

msbFirst [bool](#)

Data bit order.

testFunctional [bool](#)

Whether to log the functional result.

testValues [bool](#)

Whether to log the read results.

setup [string](#)

Optional. Setup to be applied before the pattern is run.

Details

Test Technique

- to be added

Implementation

The **Validation** section validates the test method inputs and creates the **pattern** and **pins** objects.

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies a specified `config`.

The **Body** section sets up the capture, executes the `pattern`, retrieves the functional results and data read back from the device.

The **PostBody** optionally logs the functional and parametric test records.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody. Uses the HRAM for setup and capture.

Pre Conditions

- none

Post Conditions

- none

Limitations

- Only performs reads on digital pins.

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(Pattern pattern, PinList readPins, int startIndex, int bitLength, int
wordLength, bool msbFirst, bool testFunctional,
bool testValues, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(readPins, nameof(readPins), out _pins);
        TheLib.Validate.Pattern(pattern, nameof(pattern), out _patternInfo);
        TheLib.Validate.GreaterOrEqual(startIndex, 0, nameof(startIndex));
        TheLib.Validate.GreaterOrEqual(bitLength, 1, nameof(bitLength));
        TheLib.Validate.InRange(wordLength, 1, 32, nameof(wordLength));
        _bitOrder = msbFirst ? tlBitOrder.MsbFirst : tlBitOrder.LsbFirst;
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
    }

    if (ShouldRunBody) {
        TheLib.Setup.Digital.ReadAll();
        TheLib.Execute.Digital.RunPattern(_patternInfo);
    }
}
```

```
    if (testFunctional) _patResult = TheLib.Acquire.Digital.PatternResults();
    _readWords = TheLib.Acquire.Digital.ReadWords(_pins, startIndex, bitLength,
wordLength, _bitOrder);
}

if (ShouldRunPostBody) {
    if (testFunctional) TheLib.Datalog.TestFunctional(_patResult, pattern);
    if (testValues) TheLib.Datalog.TestParametric(_readWords);
}
}
```

Class StaticPattern

Namespace: [Demo_CSRA.Functional](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class StaticPattern : TestCodeBase
```

Inheritance

[object](#) ← TestCodeBase ← StaticPattern

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
TestCodeBase.TheHdw , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
TestCodeBase.ShouldRunPreBody , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Baseline(Pattern, bool, string)

Executes a functional test with the specified pattern.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Baseline(Pattern pattern, bool testFunctional, string setup = "")
```

Parameters

pattern Pattern

The pattern to be executed during the test.

testFunctional bool

Whether to log the functional result.

setup string

Optional. Setup to be applied before the pattern is run.

Details

Test Technique

- to be added

Implementation

The **Validation** section creates the **pattern** object.

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified **config**.

The **Body** section executes the **pattern** and retrieves the results.

The **PostBody** optionally logs a functional test record.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

- none

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(Pattern pattern, bool testFunctional, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pattern(pattern, nameof(pattern), out _patternInfo);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
    }

    if (ShouldRunBody) {
        TheLib.Execute.Digital.RunPattern(_patternInfo);
        _patResult = TheLib.Acquire.Digital.PatternResults();
    }

    if (ShouldRunPostBody) {
        if (testFunctional) TheLib.Datalog.TestFunctional(_patResult, pattern);
    }
}
```

Namespace Demo_CSRA.Leakage

Classes

[Groups](#)

[Parallel](#)

[Serial](#)

Class Groups

Namespace: [Demo_CSRA.Leakage](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class Groups : TestCodeBase
```

Overview

TestClass for Groups Leakage related TestMethods.

Platform Specifics

Uses per test-instance test class object persistence ([\[TestClass\(Creation.TestInstance\)\]](#) attribute).

Inheritance

[object](#) ← TestCodeBase ← Groups

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
[TestCodeBase.TheHdw](#) , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
[TestCodeBase.ShouldRunPreBody](#) , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Baseline(PinList, double, double, double, double, string)

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Baseline(PinList pinList, double voltage, double currentRange, double
baseVoltage, double waitTime, string setup = "")
```

Parameters

pinList PinList

voltage [double](#)

currentRange [double](#)

baseVoltage [double](#)

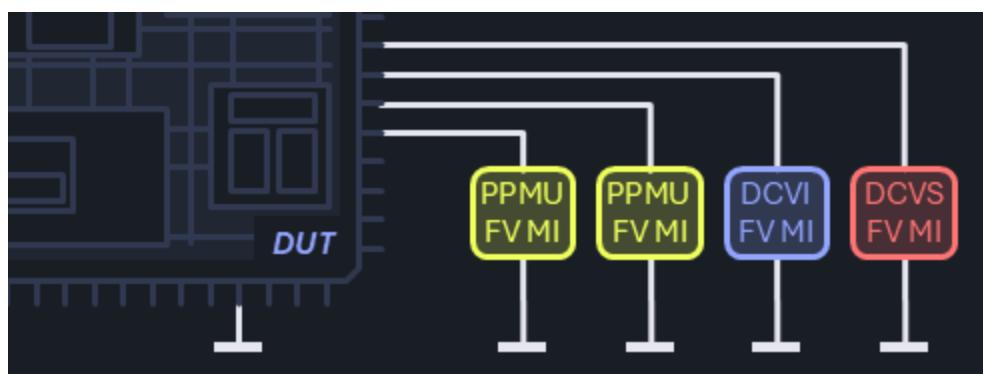
waitTime [double](#)

setup [string](#)

Details

Test Technique

This test method can be used to measure currents flowing into device inputs. One common example is the digital input leakage test to detect issues in isolation structures, possibly damaged by a manufacturing flaw or elevated test voltage levels. Even though such a device may still work according to the specification, statistical outliers can indicate early failures in the devices' target application.



The test typically applies a voltage near the VDD level to digital input pins, after the device has been brought into a static state that should not allow any current to flow. Ideally, the measured currents are very close to 0A. High sensitivity to noise, and the need to settle dynamic (charging) effects often result in significant test times, mitigated only by the fact that the measurement can usually be made on all pins in parallel.

The group testing method combines the precision of individual evaluation with the flexibility of collective assessment, allowing for the sequential analysis of each pin or group of pins, based on the structure defined in the `pinList` parameter. This approach facilitates efficient identification of leakage currents and provides a detailed overview of the behavior of the tested elements, without compromising control over each component. It is particularly useful in scenarios where pins are logically or functionally organized into groups and resources must be carefully managed. However, the testing duration may be affected by the size and complexity of the groups, as the process is carried out sequentially. In situations that require pinpoint analysis and maximum accuracy, the serial testing method is recommended. Details can be found in: Demo_CSRA->Leakage->Serial->Baseline

Conversely, for optimizing time and resource usage in less sensitive contexts, the parallel testing method remains an efficient alternative. Details can be found in: Demo_CSRA->Leakage->Parallel->Baseline

Special attention is required to avoid these common issues:

- An accidental disconnect in the signal path (e.g., due to an open DIB relay) may be difficult to detect, as measurements into an open line yield statistically inconspicuous results.
- The measurement of very small currents may be limited by the instrument's performance, so that the results rather reflect the instrument's behavior instead of the component's characteristic.

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`. For all pins specified in the `pinList` it disconnects any pin electronics, connects the dc path.

The **Body** section initially applies the `baseVoltage` and turns on the gate to all pins. Subsequently, regardless of whether individual pins or groups of pins are specified in the `pinList`, the process proceeds sequentially, element by element. For each pin or group, the `voltage` is applied, the specified `waitTime` is allowed to elapse, and a current measurement is performed. After the measurement is completed, the `baseVoltage` is reapplied to the respective pin or group.

The **PostBody** section establishes the pin electronics connection for digital pins after gating off and disconnecting the DC path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- digital pins in `pinList` have pin electronics connected

- any dc paths from pins in `pinList` are disconnected

Limitations

- none

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(PinList pinList, double voltage, double currentRange, double
baseVoltage, double waitTime, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(pinList, nameof(pinList), out _pins);
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        TheLib.Validate.MultiCondition(pinList, p => new Pins(p), nameof(_pinSteps),
out _pinSteps);
        _containsDigitalPins = _pins.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
        TheLib.Setup.Dc.Connect(_pins);
    }

    if (ShouldRunBody) {
        _meas = new();
        TheLib.Setup.Dc.SetForceAndMeter(_pins, TLibOutputMode.ForceVoltage, baseVoltage,
baseVoltage, currentRange, Measure.Current, currentRange);
        foreach (var pin in _pinSteps) {
            TheLib.Setup.Dc.ForceV(pin, voltage, currentRange, voltage, gateOn: false);
            TheLib.Execute.Wait(waitTime);
            _meas.AddRange(TheLib.Acquire.Dc.Measure(pin));
            TheLib.Setup.Dc.ForceV(pin, baseVoltage, currentRange, baseVoltage,
gateOn: false);
        }
    }

    if (ShouldRunPostBody) {
        TheLib.Setup.Dc.Disconnect(_pins);
        if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
        TheLib.Datalog.TestParametric(_meas, voltage);
    }
}
```

Preconditioning(Pattern, PinList, double, double, double, string)

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Preconditioning(Pattern pattern, PinList pinList, double voltage, double
currentRange, double baseVoltage, double waitTime, string setup = "")
```

Parameters

pattern Pattern

pinList PinList

voltage [double](#)

currentRange [double](#)

baseVoltage [double](#)

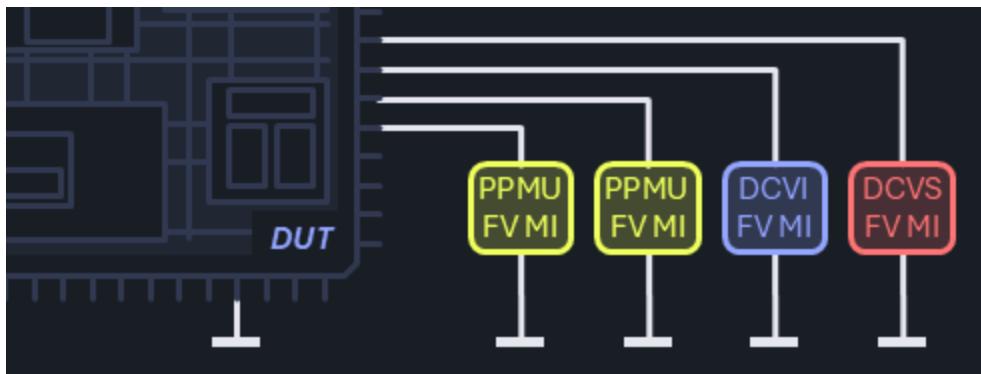
waitTime [double](#)

setup [string](#)

Details

Test Technique

This test method can be used to measure currents flowing into the inputs of a device, additionally running a pattern before the measurements to precondition the device under test (DUT) and ensure that the pins or internal circuits are in the correct state for testing. One common example is the digital input leakage test to detect issues in isolation structures, possibly damaged by a manufacturing flaw or elevated test voltage levels. Even though such a device may still work according to the specification, statistical outliers can indicate early failures in the devices' target application.



The test typically applies a voltage near the VDD level to digital input pins, after the device has been brought into a static state that should not allow any current to flow. Ideally, the measured currents are very close to 0A. High sensitivity to noise, and the need to settle dynamic (charging) effects often result in significant test times, mitigated only by the fact that the measurement can usually be made on all pins in parallel.

The group testing method combines the precision of individual evaluation with the flexibility of collective assessment, allowing for the sequential analysis of each pin or group of pins, based on the structure defined in the `pinList` parameter. This approach facilitates efficient identification of leakage currents and provides a detailed overview of the behavior of the tested elements, without compromising control over each component. It is particularly useful in scenarios where pins are logically or functionally organized into groups and resources must be carefully managed. However, the testing duration may be affected by the size and complexity of the groups, as the process is carried out sequentially. In situations that require pinpoint analysis and maximum accuracy, the serial testing method is recommended. Details can be found in: Demo_CSRA->Leakage->Serial->Preconditioning

Conversely, for optimizing time and resource usage in less sensitive contexts, the parallel testing method remains an efficient alternative. Details can be found in: Demo_CSRA->Leakage->Parallel->Preconditioning

Special attention is required to avoid these common issues:

- An accidental disconnect in the signal path (e.g., due to an open DIB relay) may be difficult to detect, as measurements into an open line yield statistically inconspicuous results.
- The measurement of very small currents may be limited by the instrument's performance, so that the results rather reflect the instrument's behavior instead of the component's characteristic.

Implementation

The **Validation** section instantiates the `pins` object and will also instantiate the `pattern` object if a pattern is given in the test method parameters.

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`. Executes the `pattern` if passed in the test method parameters and for all pins specified in the `pinList` it disconnects any pin electronics, connects the dc path.

The **Body** section initially applies the `baseVoltage` and turns on the gate to all pins. Subsequently, regardless of whether individual pins or groups of pins are specified in the `pinList`, the process proceeds sequentially, element by element. For each pin or group, the `voltage` is applied, the specified `waitTime` is allowed to elapse, and a current measurement is performed. After the measurement is completed, the `baseVoltage` is reapplied to the respective pin or group.

The **PostBody** section establishes the pin electronics connection for digital pins after gating off and disconnecting the DC path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- digital pins in `pinList` have pin electronics connected
- any dc paths from pins in `pinList` are disconnected

Limitations

- none

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Preconditioning(Pattern pattern, PinList pinList, double voltage, double
currentRange, double baseVoltage, double waitTime,
    string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(pinList, nameof(pinList), out _pins);
        TheLib.Validate.Pattern(pattern, nameof(pattern), out _pattern);
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        TheLib.Validate.MultiCondition(pinList, p => new Pins(p), nameof(_pinSteps),
            out _pinSteps);
        _containsDigitalPins = _pins.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        TheLib.Execute.Digital.RunPattern(_pattern);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
        TheLib.Setup.Dc.Connect(_pins);
    }
}
```

```
}

if (ShouldRunBody) {
    _meas = new();
    TheLib.Setup.Dc.SetForceAndMeter(_pins, TLibOutputMode.ForceVoltage, baseVoltage,
baseVoltage, currentRange, Measure.Current, currentRange);
    foreach (var pin in _pinSteps) {
        TheLib.Setup.Dc.ForceV(pin, voltage, currentRange, voltage, gateOn: false);
        TheLib.Execute.Wait(waitTime);
        _meas.AddRange(TheLib.Acquire.Dc.Measure(pin));
        TheLib.Setup.Dc.ForceV(pin, baseVoltage, currentRange, baseVoltage,
gateOn: false);
    }
}

if (ShouldRunPostBody) {
    TheLib.Setup.Dc.Disconnect(_pins);
    if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
    TheLib.Datalog.TestParametric(_meas, voltage);
}
}
```

Class Parallel

Namespace: [Demo_CSRA.Leakage](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class Parallel : TestCodeBase
```

Overview

TestClass for Parallel Leakage related TestMethods.

Platform Specifics

Uses per test-instance test class object persistence ([\[TestClass\(Creation.TestInstance\)\]](#) attribute).

Inheritance

[object](#) ← TestCodeBase ← Parallel

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
[TestCodeBase.TheHdw](#) , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
[TestCodeBase.ShouldRunPreBody](#) , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

[Baseline\(PinList, double, double, double, string\)](#)

Measures leakage currents by applying bias voltage to all pins simultaneously, in a parallel testing process.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Baseline(PinList pinList, double voltage, double currentRange, double waitTime,
string setup = "")
```

Parameters

pinList PinList

List of pin or pin group names.

voltage [double](#)

The force voltage value.

currentRange [double](#)

The current range for measurement.

waitTime [double](#)

The settling time before the measurement.

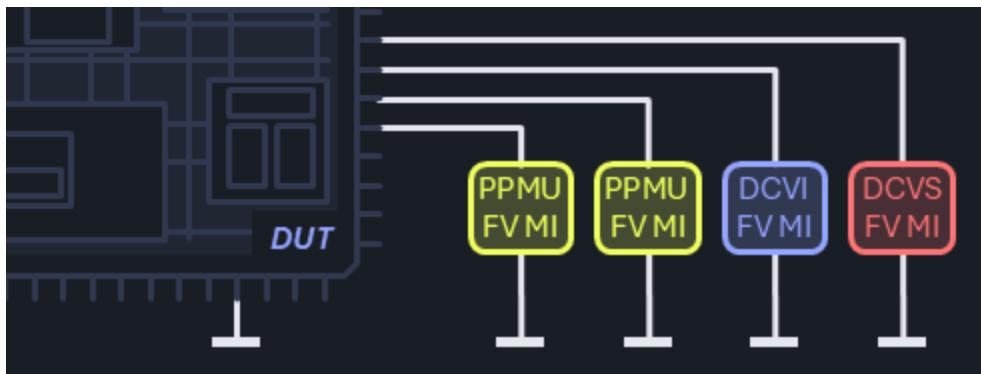
setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

This test method can be used to measure currents flowing into device inputs. One common example is the digital input leakage test to detect issues in isolation structures, possibly damaged by a manufacturing flaw or elevated test voltage levels. Even though such a device may still work according to the specification, statistical outliers can indicate early failures in the devices' target application.



The test typically applies a voltage near the VDD level to digital input pins, after the device has been brought into a static state that should not allow any current to flow. Ideally, the measured currents are very close to 0A. High sensitivity to noise, and the need to settle dynamic (charging) effects often result in significant test times, mitigated only by the fact that the measurement can usually be made on all pins in parallel.

The parallel testing method stands out for its high level of efficiency, as it allows the simultaneous evaluation of multiple pins, thereby significantly reducing testing time and optimizing the use of available resources. However, a major limitation of this approach lies in its inability to detect leakage currents between input pins, since all pins are tested concurrently. In such cases, it is recommended to resort to the serial testing method, which enables a more precise and individual analysis of each pin. Details can be found in: Demo_CSRA->Leakage->Serial->Baseline

Alternatively, the group testing method offers a balance between precision and efficiency by performing sequential testing of pins or groups of pins, according to the structure defined in the `pinList`. Details can be found in: Demo_CSRA->Leakage->Groups->Baseline

Special attention is required to avoid these common issues:

- An accidental disconnect in the signal path (e.g., due to an open DIB relay) may be difficult to detect, as measurements into an open line yield statistically inconspicuous results.
- The measurement of very small currents may be limited by the instrument's performance, so that the results rather reflect the instrument's behavior instead of the component's characteristic.

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`. For all pins specified in the `pinList` it disconnects any pin electronics and connects the dc path.

The **Body** section applies a force `voltage`, turns on the gate to all pins, and performs a current measurement on all pins in parallel after the specified `waitTime`.

The **PostBody** section establishes the pin electronics connection for digital pins after gating off and disconnecting the DC path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- digital pins in `pinList` have pin electronics connected
- any dc paths from pins in `pinList` are disconnected

Limitations

- none

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(PinList pinList, double voltage, double currentRange, double waitTime,
string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(pinList, nameof(pinList), out _pins);
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        _containsDigitalPins = _pins.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
        TheLib.Setup.Dc.Connect(_pins);
    }

    if (ShouldRunBody) {
        TheLib.Setup.Dc.SetForceAndMeter(_pins, TLibOutputMode.ForceVoltage, voltage,
voltage, currentRange, Measure.Current, currentRange);
        TheLib.Execute.Wait(waitTime);
        _meas = TheLib.Acquire.Dc.Measure(_pins);
    }

    if (ShouldRunPostBody) {
        TheLib.Setup.Dc.Disconnect(_pins);
        if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
        TheLib.Datalog.TestParametric(_meas, voltage);
    }
}
```

```
    }  
}
```

Preconditioning(Pattern, PinList, double, double, double, string)

Runs a pattern and then measures leakage currents by applying bias voltage to all pins simultaneously, using a parallel testing process.

```
[TestMethod]  
[Steppable]  
[CustomValidation]  
public void Preconditioning(Pattern pattern, PinList pinList, double voltage, double  
currentRange, double waitTime, string setup = "")
```

Parameters

pattern Pattern

Pattern to be executed.

pinList PinList

List of pin or pin group names.

voltage [double](#)

The force voltage value.

currentRange [double](#)

The current range for measurement.

waitTime [double](#)

The settling time before the measurement.

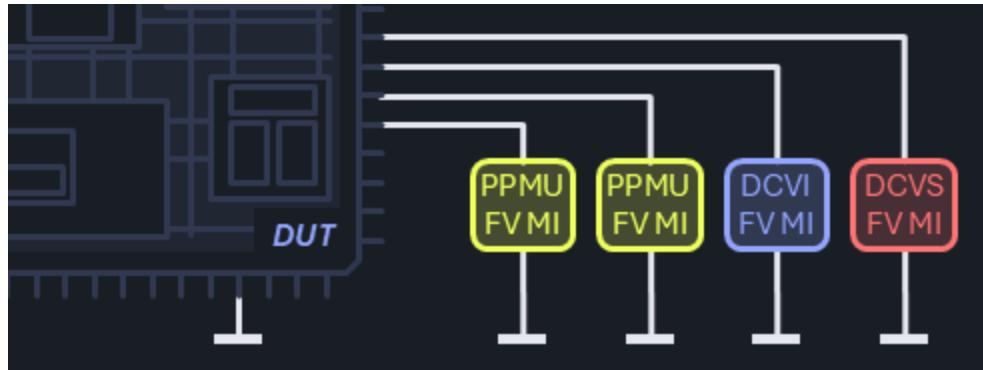
setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

This test method can be used to measure currents flowing into the inputs of a device, additionally running a pattern before the measurements to precondition the device under test (DUT) and ensure that the pins or internal circuits are in the correct state for testing. One common example is the digital input leakage test to detect issues in isolation structures, possibly damaged by a manufacturing flaw or elevated test voltage levels. Even though such a device may still work according to the specification, statistical outliers can indicate early failures in the devices' target application.



The test typically applies a voltage near the VDD level to digital input pins, after the device has been brought into a static state that should not allow any current to flow. Ideally, the measured currents are very close to 0A. High sensitivity to noise, and the need to settle dynamic (charging) effects often result in significant test times, mitigated only by the fact that the measurement can usually be made on all pins in parallel.

The parallel testing method stands out for its high level of efficiency, as it allows the simultaneous evaluation of multiple pins, thereby significantly reducing testing time and optimizing the use of available resources. However, a major limitation of this approach lies in its inability to detect leakage currents between input pins, since all pins are tested concurrently. In such cases, it is recommended to resort to the serial testing method, which enables a more precise and individual analysis of each pin. Details can be found in: Demo_CSRA->Leakage->Serial->Preconditioning

Alternatively, the group testing method offers a balance between precision and efficiency by performing sequential testing of pins or groups of pins, according to the structure defined in the [pinList](#). Details can be found in: Demo_CSRA->Leakage->Groups->Preconditioning

Special attention is required to avoid these common issues:

- An accidental disconnect in the signal path (e.g., due to an open DIB relay) may be difficult to detect, as measurements into an open line yield statistically inconspicuous results.
- The measurement of very small currents may be limited by the instrument's performance, so that the results rather reflect the instrument's behavior instead of the component's characteristic.

Implementation

The **Validation** section instantiates the pins object and will also instantiate the pattern object if a pattern is given in the test method parameters.

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`. Executes the `pattern` if passed in the test method parameters and for all pins specified in the `pinList` it disconnects any pin electronics, connects the dc path.

The **Body** section applies a force `voltage`, turns on the gate to all pins, and performs a current measurement on all pins in parallel after the specified `waitTime`.

The **PostBody** section establishes the pin electronics connection for digital pins after gating off and disconnecting the DC path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- digital pins in `pinList` have pin electronics connected
- any dc paths from pins in `pinList` are disconnected

Limitations

- none

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Preconditioning(Pattern pattern, PinList pinList, double voltage, double
currentRange, double waitTime, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(pinList, nameof(pinList), out _pins);
        TheLib.Validate.Pattern(pattern, nameof(pattern), out _pattern);
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        _containsDigitalPins = _pins.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        TheLib.Execute.Digital.RunPattern(_pattern);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
        TheLib.Setup.Dc.Connect(_pins);
    }
}
```

```
if (ShouldRunBody) {
    TheLib.Setup.Dc.SetForceAndMeter(_pins, TLibOutputMode.ForceVoltage, voltage,
voltage, currentRange, Measure.Current, currentRange);
    TheLib.Execute.Wait(waitTime);
    _meas = TheLib.Acquire.Dc.Measure(_pins);
}

if (ShouldRunPostBody) {
    TheLib.Setup.Dc.Disconnect(_pins);
    if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
    TheLib.Datalog.TestParametric(_meas, voltage);
}
}
```

Class Serial

Namespace: [Demo_CSRA.Leakage](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class Serial : TestCodeBase
```

Overview

TestClass for Serial Leakage related TestMethods.

Platform Specifics

Uses per test-instance test class object persistence ([\[TestClass\(Creation.TestInstance\)\]](#) attribute).

Inheritance

[object](#) ← TestCodeBase ← Serial

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
[TestCodeBase.TheHdw](#) , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
[TestCodeBase.ShouldRunPreBody](#) , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Baseline(PinList, double, double, double, double, string)

Measures leakage currents by applying bias voltage to each pin in a serial testing process.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Baseline(PinList pinList, double voltage, double currentRange, double
baseVoltage, double waitTime, string setup = "")
```

Parameters

pinList [PinList](#)

List of pin or pin group names.

voltage [double](#)

The force voltage value.

currentRange [double](#)

The current range for measurement.

baseVoltage [double](#)

The voltage value applied to all pins other than the one currently measured in the sequence. Intended to prevent any cross-pin interference.

waitTime [double](#)

The settling time before the measurement.

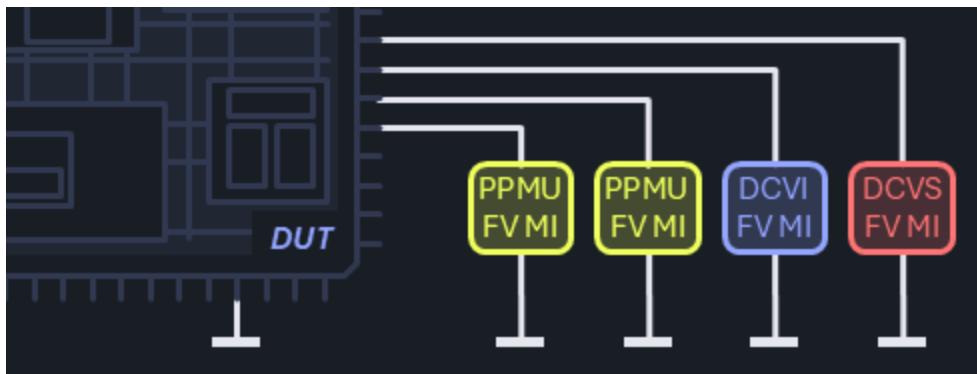
setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

This test method can be used to measure currents flowing into device inputs. One common example is the digital input leakage test to detect issues in isolation structures, possibly damaged by a manufacturing flaw or elevated test voltage levels. Even though such a device may still work according to the specification, statistical outliers can indicate early failures in the devices' target application.



The test typically applies a voltage near the VDD level to digital input pins, after the device has been brought into a static state that should not allow any current to flow. Ideally, the measured currents are very close to 0A. High sensitivity to noise, and the need to settle dynamic (charging) effects often result in significant test times, mitigated only by the fact that the measurement can usually be made on all pins in parallel.

The serial testing method is distinguished by a high degree of precision, as it allows for the individual evaluation of each pin, thereby contributing to the efficient identification of leakage currents and a detailed analysis of the behavior of each tested element. It is also ideal in cases where resources are shared. However, a significant limitation of this method lies in the longer testing time, since the evaluation is performed sequentially, pin by pin. In such situations, it is recommended to resort to the parallel testing method, which enables faster testing and more efficient use of available resources.

Details can be found in: Demo_CSRA->Leakage->Parallel->Baseline

Alternatively, the group testing method offers a balance between precision and efficiency by performing sequential testing of pins or groups of pins, according to the structure defined in the `pinList`. Details can be found in: Demo_CSRA->Leakage->Groups->Baseline

Special attention is required to avoid these common issues:

- An accidental disconnect in the signal path (e.g., due to an open DIB relay) may be difficult to detect, as measurements into an open line yield statistically inconspicuous results.
- The measurement of very small currents may be limited by the instrument's performance, so that the results rather reflect the instrument's behavior instead of the component's characteristic.

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`. For all pins specified in the `pinList` it disconnects any pin electronics, connects the dc path.

The **Body** section initially applies the `baseVoltage` and turns on the gate to all pins. Subsequently, regardless of whether a group of pins or multiple groups are specified, the process proceeds sequentially, pin by pin. For each individual pin, the `voltage` is applied, the specified `waitTime` is allowed

to elapse, and a current measurement is performed. After the measurement is completed, the `baseVoltage` is reapplied to the respective pin.

The **PostBody** section establishes the pin electronics connection for digital pins after gating off and disconnecting the DC path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- digital pins in `pinList` have pin electronics connected
- any dc paths from pins in `pinList` are disconnected

Limitations

- none

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(PinList pinList, double voltage, double currentRange, double
baseVoltage, double waitTime, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(pinList, nameof(pinList), out _pins);
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        _pinSteps = _pins.Select(pin => new Pins(pin.Name)).ToArray();
        _containsDigitalPins = _pins.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
        TheLib.Setup.Dc.Connect(_pins);
    }

    if (ShouldRunBody) {
        _meas = new();
        TheLib.Setup.Dc.SetForceAndMeter(_pins, TLibOutputMode.ForceVoltage, baseVoltage,
baseVoltage, currentRange, Measure.Current, currentRange);
        foreach (var pin in _pinSteps) {
```

```

        TheLib.Setup.Dc.ForceV(pin, voltage, currentRange, voltage, gateOn: false);
        TheLib.Execute.Wait(waitTime);
        _meas.AddRange(TheLib.Acquire.Dc.Measure(pin));
        TheLib.Setup.Dc.ForceV(pin, baseVoltage, currentRange, baseVoltage,
gateOn: false);
    }
}

if (ShouldRunPostBody) {
    TheLib.Setup.Dc.Disconnect(_pins);
    if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
    TheLib.Datalog.TestParametric(_meas, voltage);
}
}

```

Preconditioning(Pattern, PinList, double, double, double, string)

Runs a pattern and then measures leakage currents by applying bias voltage to each pin in a serial testing process.

```

[TestMethod]
[Steppable]
[CustomValidation]
public void Preconditioning(Pattern pattern, PinList pinList, double voltage, double
currentRange, double baseVoltage, double waitTime, string setup = "")

```

Parameters

pattern Pattern

Pattern to be executed.

pinList PinList

List of pin or pin group names.

voltage [double](#)

The force voltage value.

currentRange [double](#)

The current range for measurement.

baseVoltage [double](#)

The voltage value applied to all pins other than the one currently measured in the sequence. Intended to prevent any cross-pin interference.

waitTime [double](#)

The settling time before the measurement.

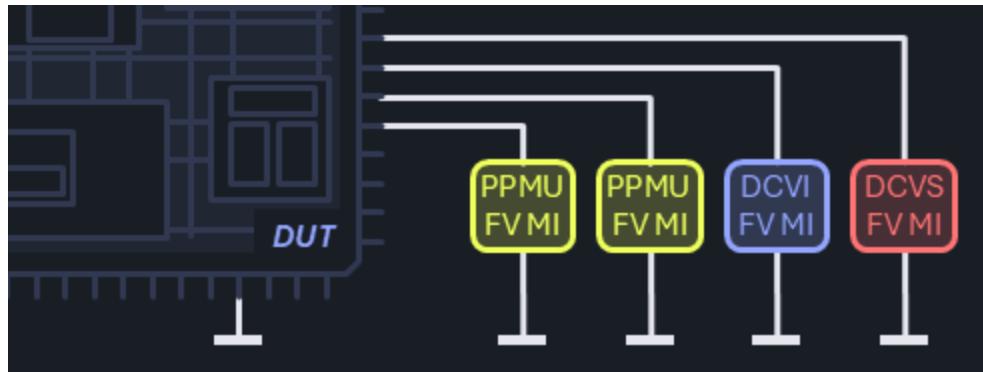
setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

This test method can be used to measure currents flowing into the inputs of a device, additionally running a pattern before the measurements to precondition the device under test (DUT) and ensure that the pins or internal circuits are in the correct state for testing. One common example is the digital input leakage test to detect issues in isolation structures, possibly damaged by a manufacturing flaw or elevated test voltage levels. Even though such a device may still work according to the specification, statistical outliers can indicate early failures in the devices' target application.



The test typically applies a voltage near the VDD level to digital input pins, after the device has been brought into a static state that should not allow any current to flow. Ideally, the measured currents are very close to 0A. High sensitivity to noise, and the need to settle dynamic (charging) effects often result in significant test times, mitigated only by the fact that the measurement can usually be made on all pins in parallel.

The serial testing method is distinguished by a high degree of precision, as it allows for the individual evaluation of each pin, thereby contributing to the efficient identification of leakage currents and a detailed analysis of the behavior of each tested element. It is also ideal in cases where resources are shared. However, a significant limitation of this method lies in the longer testing time, since the

evaluation is performed sequentially, pin by pin. In such situations, it is recommended to resort to the parallel testing method, which enables faster testing and more efficient use of available resources. Details can be found in: Demo_CSRA->Leakage->Parallel->Preconditioning

Alternatively, the group testing method offers a balance between precision and efficiency by performing sequential testing of pins or groups of pins, according to the structure defined in the `pinList`. Details can be found in: Demo_CSRA->Leakage->Groups->Preconditioning

Special attention is required to avoid these common issues:

- An accidental disconnect in the signal path (e.g., due to an open DIB relay) may be difficult to detect, as measurements into an open line yield statistically inconspicuous results.
- The measurement of very small currents may be limited by the instrument's performance, so that the results rather reflect the instrument's behavior instead of the component's characteristic.

Implementation

The **Validation** section instantiates the `pins` object and will also instantiate the `pattern` object if a pattern is given in the test method parameters.

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`. Executes the `pattern` if passed in the test method parameters and for all pins specified in the `pinList` it disconnects any pin electronics, connects the dc path.

The **Body** section initially applies the `baseVoltage` and turns on the gate to all pins. Subsequently, regardless of whether a group of pins or multiple groups are specified, the process proceeds sequentially, pin by pin. For each individual pin, the `voltage` is applied, the specified `waitTime` is allowed to elapse, and a current measurement is performed. After the measurement is completed, the `baseVoltage` is reapplied to the respective pin.

The **PostBody** section establishes the pin electronics connection for digital pins after gating off and disconnecting the DC path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- digital pins in `pinList` have pin electronics connected
- any dc paths from pins in `pinList` are disconnected

Limitations

- none

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Preconditioning(Pattern pattern, PinList pinList, double voltage, double
currentRange, double baseVoltage, double waitTime,
    string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(pinList, nameof(pinList), out _pins);
        TheLib.Validate.Pattern(pattern, nameof(pattern), out _pattern);
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        _pinSteps = _pins.Select(pin => new Pins(pin.Name)).ToArray();
        _containsDigitalPins = _pins.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        TheLib.Execute.Digital.RunPattern(_pattern);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
        TheLib.Setup.Dc.Connect(_pins);
    }

    if (ShouldRunBody) {
        _meas = new();
        TheLib.Setup.Dc.SetForceAndMeter(_pins, TLibOutputMode.ForceVoltage, baseVoltage,
baseVoltage, currentRange, Measure.Current, currentRange);
        foreach (var pin in _pinSteps) {
            TheLib.Setup.Dc.ForceV(pin, voltage, currentRange, voltage, gateOn: false);
            TheLib.Execute.Wait(waitTime);
            _meas.AddRange(TheLib.Acquire.Dc.Measure(pin));
            TheLib.Setup.Dc.ForceV(pin, baseVoltage, currentRange, baseVoltage,
gateOn: false);
        }
    }

    if (ShouldRunPostBody) {
        TheLib.Setup.Dc.Disconnect(_pins);
        if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
        TheLib.Datalog.TestParametric(_meas, voltage);
    }
}
```

Namespace Demo_CSRA.Parametric

Overview

TestClass for all generic parametric TestMethods.

Platform Specifics

Uses per test-instance test class object persistence (`[TestClass(Creation.TestInstance)]` attribute).

Classes

[MultiCondition](#)

[PatternHandshake](#)

[SingleCondition](#)

Class MultiCondition

Namespace: [Demo_CSRA.Parametric](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class MultiCondition : TestCodeBase
```

Overview

TestClass for all generic parametric TestMethods using diverse (multi) conditions.

Platform Specifics

Uses per test-instance test class object persistence ([\[TestClass\(Creation.TestInstance\)\]](#) attribute).

Inheritance

[object](#) ← TestCodeBase ← MultiCondition

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
[TestCodeBase.TheHdw](#) , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
[TestCodeBase.ShouldRunPreBody](#) , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

[Baseline\(PinList, string, string, string, string, string, string, PinList, double, string\)](#)

Parametric measurement by setting up all force Pins, then measuring all force or optionally different measure Pins.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Baseline(PinList forcePinList, string forceModes, string forceValues, string
clampValues, string measureModes, string measureRanges, string sampleSizes = "1", PinList
measPinList = null, double waitTime = 0, string setup = "")
```

Parameters

forcePinList PinList

Comma separated list of pin or pin groups representing the DC setup and/or measurement.

forceModes [string](#)

Comma separated list of the force modes for each pin or pin group.

forceValues [string](#)

Comma separated list of force voltages or currents for each pin or pin group.

clampValues [string](#)

Comma separated list of clamp voltages or currents for each pin or pin group.

measureModes [string](#)

Comma separated list of the measure modes for each pin or pin group.

measureRanges [string](#)

Comma separated list of the measure ranges for each pin or pin group.

sampleSizes [string](#)

Optional. Comma separated list of number of samples to average for each pin or pin group.

measPinList PinList

Optional. Comma separated list of measurement pin or pin groups, if different from forcePinList.

waitTime [double](#)

Optional. The wait time before the measurement.

setup [string](#)

Optional. Setup set to configure the dib or device.

Details

Test Technique

- to be added

Implementation

The **Validation** section creates the [Pins](#) objects, converts the comma separated value (CSV) lists into cached value arrays, and determines force & measure modes and confirms valid combinations.

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified [setup](#). For all pins specified in the [pinList](#) it disconnects any pin electronics, connects the dc path and turns on the gate.

The **Body** section applies the force condition(s) on all pins, sets the measurement mode(s) and performs the measurement on all pins in parallel after the specified [waitTime](#).

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(PinList forcePinList, string forceModes, string forceValues, string
clampValues, string measureModes, string measureRanges,
string sampleSizes = "1", PinList measPinList = null, double waitTime = 0, string setup
```

```

= "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(forcePinList, nameof(forcePinList), out _pins);
        _containsDigitalPins = _pins.ContainsFeature(InstrumentFeature.Digital);
        TheLib.Validate.MultiCondition(forcePinList, p => new Pins(p),
nameof(_pinsForceGroups), out _pinsForceGroups);
        if (string.IsNullOrEmpty(measPinList)) _pinsMeasureGroups = _pinsForceGroups;
        else {
            TheLib.Validate.Pins(measPinList, nameof(measPinList), out _);
            TheLib.Validate.MultiCondition(measPinList, p => new Pins(p),
nameof(_pinsMeasureGroups), out _pinsMeasureGroups);
            _pins.Add(measPinList);
        }
        TheLib.Validate.MultiCondition(forceModes, nameof(forceModes), out
_outputModes, _pinsForceGroups.Length);
        TheLib.Validate.MultiCondition(forceValues, double.Parse, nameof(forceValues), out
_forceValues, _pinsForceGroups.Length);
        TheLib.Validate.MultiCondition(clampValues, double.Parse, nameof(clampValues), out
_clampValues, _pinsForceGroups.Length);
        TheLib.Validate.MultiCondition(measureModes, nameof(measureModes), out
_measureModes, _pinsMeasureGroups.Length);
        TheLib.Validate.MultiCondition(measureRanges, double.Parse, nameof(measureRanges),
out _measureRanges, _pinsMeasureGroups.Length);
        TheLib.Validate.MultiCondition(sampleSizes, int.Parse, nameof(sampleSizes), out
_sampleSizes, _pinsMeasureGroups.Length);
    }
    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
        TheLib.Setup.Dc.Connect(_pins);
    }

    if (ShouldRunBody) {
        TheLib.Setup.Dc.Force(_pinsForceGroups, _outputModes, _forceValues, _forceValues,
_clampValues, [true]);
        TheLib.Setup.Dc.SetMeter(_pinsMeasureGroups, _measureModes, _measureRanges);
        TheLib.Execute.Wait(waitTime);
        _measureValues = TheLib.Acquire.Dc.Measure(_pinsMeasureGroups, _sampleSizes);
    }

    if (ShouldRunPostBody) {
        TheLib.Setup.Dc.Disconnect(_pins);
        if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
        TheLib.Datalog.TestParametric(_measureValues);
    }
}

```

```
    }  
}
```

PreconditionPattern(PinList, string, string, string, Pattern, string, string, string, PinList, double, string)

Parametric measurement by running preconditioning pattern, setting up force Pins and then measuring all force or optionally different measure Pins.

```
[TestMethod]  
[Steppable]  
[CustomValidation]  
public void PreconditionPattern(PinList forcePinList, string forceModes, string forceValues,  
string clampValues, Pattern preconditionPat, string measureModes, string measureRanges,  
string sampleSizes = "1", PinList measPinList = null, double waitTime = 0, string setup  
= "")
```

Parameters

forcePinList PinList

Comma separated list of pin or pin groups representing the DC setup and/or measurement.

forceModes [string](#)

Comma separated list of the force modes for each pin or pin group.

forceValues [string](#)

Comma separated list of force voltages or currents for each pin or pin group.

clampValues [string](#)

Comma separated list of clamp voltages or currents for each pin or pin group.

preconditionPat Pattern

Pattern to run to precondition the device before the parametric test.

measureModes [string](#)

Comma separated list of the measure modes for each pin or pin group.

measureRanges [string](#)

Comma separated list of the measure ranges for each pin or pin group.

sampleSizes [string](#)

Optional. Comma separated list of number of samples to average for each pin or pin group.

measPinList [PinList](#)

Optional. Comma separated list of measurement pin or pin groups, if different from forcePinList.

waitTime [double](#)

Optional. The wait time before the measurement.

setup [string](#)

Optional. Setup set to configure the dib or device.

Details

Test Technique

- to be added

Implementation

The **Validation** section creates the **Pins** and **PatternInfo** objects, converts the comma separated value (CSV) lists into cached value arrays, and determines force & measure modes and confirms valid combinations.

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified **setup**. The preconditioning pattern is executed. For all pins specified in the **pinList** it disconnects any pin electronics, connects the dc path and turns on the gate.

The **Body** section applies the force condition(s) on all pins, sets the measurement mode(s) and performs the measurement on all pins in parallel after the specified **waitTime**.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void PreconditionPattern(PinList forcePinList, string forceModes, string forceValues,
string clampValues, Pattern preconditionPat,
    string measureModes, string measureRanges, string sampleSizes = "1", PinList measPinList
= null, double waitTime = 0, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(forcePinList, nameof(forcePinList), out _pins);
        _containsDigitalPins = _pins.ContainsFeature(InstrumentFeature.Digital);
        TheLib.Validate.MultiCondition(forcePinList, p => new Pins(p), nameof(forcePinList),
out _pinsForceGroups);
        if (string.IsNullOrEmpty(measPinList)) _pinsMeasureGroups = _pinsForceGroups;
        else {
            TheLib.Validate.Pins(measPinList, nameof(measPinList), out _);
            TheLib.Validate.MultiCondition(measPinList, p => new Pins(p),
nameof(measPinList), out _pinsMeasureGroups);
            _pins.Add(measPinList);
        }
        _patternSpecified = !string.IsNullOrEmpty(preconditionPat);
        if (_patternSpecified) TheLib.Validate.Pattern(preconditionPat,
nameof(preconditionPat), out _pattern);
        else Services.Alert.Error("Invalid Pattern. The pattern is null or empty.");
        TheLib.Validate.MultiCondition(forceModes, nameof(forceModes), out
_outputModes, _pinsForceGroups.Length);
        TheLib.Validate.MultiCondition(forceValues, double.Parse, nameof(forceValues), out
_forceValues, _pinsForceGroups.Length);
        TheLib.Validate.MultiCondition(clampValues, double.Parse, nameof(clampValues), out
_clampValues, _pinsForceGroups.Length);
        TheLib.Validate.MultiCondition(measureModes, nameof(measureModes), out
_measureModes, _pinsMeasureGroups.Length);
        TheLib.Validate.MultiCondition(measureRanges, double.Parse, nameof(measureRanges),
out _measureRanges, _pinsMeasureGroups.Length);
        TheLib.Validate.MultiCondition(sampleSizes, int.Parse, nameof(sampleSizes), out
_sampleSizes, _pinsMeasureGroups.Length);
    }
}
```

```
if (ShouldRunPreBody) {
    TheLib.Setup.LevelsAndTiming.Apply(true);
    Services.Setup.Apply(setup);
    if (_patternSpecified) TheLib.Execute.Digital.RunPattern(_pattern);
    if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
    TheLib.Setup.Dc.Connect(_pins);
}

if (ShouldRunBody) {
    TheLib.Setup.Dc.Force(_pinsForceGroups, _outputModes, _forceValues, _forceValues,
    _clampValues, [true]);
    TheLib.Setup.Dc.SetMeter(_pinsMeasureGroups, _measureModes, _measureRanges);
    TheLib.Execute.Wait(waitTime);
    _measureValues = TheLib.Acquire.Dc.Measure(_pinsMeasureGroups, _sampleSizes);
}

if (ShouldRunPostBody) {
    TheLib.Setup.Dc.Disconnect(_pins);
    if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
    TheLib.Datalog.TestParametric(_measureValues);
}
}
```

Class PatternHandshake

Namespace: [Demo_CSRA.Parametric](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class PatternHandshake : TestCodeBase
```

Overview

TestClass for all pattern handshaking related TestMethods.

Platform Specifics

Uses per test-instance test class object persistence ([\[TestClass\(Creation.TestInstance\)\]](#) attribute).

Inheritance

[object](#) ← TestCodeBase ← PatternHandshake

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
[TestCodeBase.TheHdw](#) , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
[TestCodeBase.ShouldRunPreBody](#) , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Baseline(Pattern, int, int, string, bool, string)

Runs the specified pattern and executes the stopAction at each occurrence of stopFlag in the pattern. Every value returned from the stopAction(s) will be datalogged.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Baseline(Pattern pattern, int stopFlag, int numberOfStops, string stopAction,
bool testFunctional, string setup = "")
```

Parameters

pattern Pattern

Pattern name to be executed.

stopFlag [int](#)

Pattern flag to stop at.

numberOfStops [int](#)

Number of total stop in the pattern.

stopAction [string](#)

Action to be called at each stop.

testFunctional [bool](#)

Whether to test the functional results.

setup [string](#)

Optional. Action to configure the dib or device.

Details

Test Technique

- to be added

Implementation

The **Validation** section instantiates the **pattern** object and converts the **stopAction** parameter into a delegate that can be called. In order for this conversion to occur, there must be a function in the loaded dll's that is the same name as the **stopAction** parameter and has the prototype:

```
public static List<PinSite<double>> name(PatternInfo pattern, int stops);
```

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `config`.

The **Body** section executes the RunPatternConditionalStop method and stores the results in values. The RunPatternConditionalStop method performs the following:

1. Starts the `pattern`
2. Waits for the cpu flag condition to occur defined by the `stopFlag` parameter
3. Executes the `stopAction` delegate function, passing in the `pattern` and the current stop index.
4. Adds the results returned from the `stopAction` delegate to its return value
5. Continues to the next pattern stop by clearing the `stopFlag`
6. Repeat steps 2-5 for each stop in the `pattern`

All cpu flags that are present in the pattern must be properly handled during the RunPatternConditionalStop method or the patgen will hit a timeout. It is possible to have different flag conditions for each stop. These can be updated by the `stopAction` delegate through the `pattern` SetFlag parameter.

The **PostBody** conditionally logs the functional test record and logs each parametric result returned by the `stopAction` delegate.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

- none

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(Pattern pattern, int stopFlag, int numberOfStops, string stopAction,
bool testFunctional, string setup = "") {

    if (TheExec.Flow.IsValidating) {
```

```

        TheLib.Validate.Pattern(pattern, nameof(pattern), out _pattern);
        TheLib.Validate.GreaterOrEqual(stopFlag, 0, nameof(stopFlag));
        TheLib.Validate.GreaterOrEqual(numberOfStops, 0, nameof(numberOfStops));
        _pattern.SetFlags = stopFlag;
        TheLib.Validate.MethodHandle(stopAction, nameof(stopAction), out _stopAction);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
    }

    if (ShouldRunBody) {
        _values = TheLib.Execute.Digital.RunPatternConditionalStop(_pattern,
numberOfStops, _stopAction.Execute);
        if (testFunctional) _patResult = TheLib.Acquire.Digital.PatternResults();
    }

    if (ShouldRunPostBody) {
        if (testFunctional) TheLib.Datalog.TestFunctional(_patResult, pattern);
        foreach (var value in _values) {
            TheLib.Datalog.TestParametric(value);
        }
    }
}

```

ExampleAction(PatternInfo, int)

```
public static List<PinSite<double>> ExampleAction(PatternInfo pattern, int stop)
```

Parameters

pattern [PatternInfo](#)

stop [int](#)

Returns

[List](#)<PinSite<[double](#)>>

Class SingleCondition

Namespace: [Demo_CSRA.Parametric](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class SingleCondition : TestCodeBase
```

Overview

TestClass for all generic parametric TestMethods using uniform (single) conditions.

Platform Specifics

Uses per test-instance test class object persistence ([\[TestClass\(Creation.TestInstance\)\]](#) attribute).

Inheritance

[object](#) ← TestCodeBase ← SingleCondition

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
[TestCodeBase.TheHdw](#) , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
[TestCodeBase.ShouldRunPreBody](#) , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

[Baseline\(PinList, string, double, double, string, double, int, PinList, double, string\)](#)

Parametric measurement by setting up all force Pins, then measuring all force or optionally different measure Pins.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Baseline(PinList forcePinList, string forceMode, double forceValue, double
clampValue, string measureWhat, double measureRange, int sampleSize = 1, PinList measPinList
= null, double waitTime = 0, string setup = "")
```

Parameters

forcePinList PinList

Comma separated list of pin or pin groups representing the DC setup and/or measurement.

forceMode [string](#)

Force mode for each pin or pin group.

forceValue [double](#)

Force voltage or current for all pins or pin groups.

clampValue [double](#)

Clamp voltage or current for all pina or pin groups.

measureWhat [string](#)

Measure either voltage or current for all measure pins or pin groups.

measureRange [double](#)

Expected voltage or current for all pins or pin groups to set the range.

sampleSize [int](#)

Optional. Number of samples to average for all pins or pin groups.

measPinList PinList

Optional. Comma separted list of measurement pins or pin groups, if different from forcePinList.

waitTime [double](#)

Optional. Settling time used after pin setup.

setup [string](#)

Optional. Setup setting to preconfigure the dib or device.

Details

Test Technique

- to be added

Implementation

The **Validation** section creates the [Pins](#) objects, determines force & measure modes and confirms valid combinations.

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified [setup](#). For all pins specified in the [pinList](#) it disconnects any pin electronics, connects the dc path and turns on the gate.

The **Body** section applies the force condition on all pins, sets the measurement mode and performs the measurement on all pins in parallel after the specified [waitTime](#).

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(PinList forcePinList, string forceMode, double forceValue, double
clampValue, string measureWhat, double measureRange,
int sampleSize = 1, PinList measPinList = null, double waitTime = 0.0, string setup =
```

```

"") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(forcePinList, nameof(forcePinList), out _pinsForce);
        TheLib.Validate.Pins(forcePinList, nameof(forcePinList), out _pinsAll);
        _measPinListIsNullOrEmpty = string.IsNullOrEmpty(measPinList);
        if (_measPinListIsNullOrEmpty) {
            _pinsMeasure = new Pins(forcePinList);
        } else {
            TheLib.Validate.Pins(measPinList, nameof(measPinList), out _pinsMeasure);
            _pinsAll.Add(measPinList);
        }
        TheLib.Validate.Enum(forceMode, nameof(forceMode), out _outputMode);
        TheLib.Validate.Enum(measureWhat, nameof(measureWhat), out _measureMode);
        _outputRangeValue = (_outputMode == TLibOutputMode.ForceVoltage) ? clampValue
        : forceValue;
        _containsDigitalPins = _pinsAll.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pinsAll);
        TheLib.Setup.Dc.Connect(_pinsAll);
    }

    if (ShouldRunBody) {
        TheLib.Setup.Dc.Force(_pinsForce, _outputMode, forceValue, forceValue, clampValue);
        TheLib.Setup.Dc.SetMeter(_pinsMeasure, _measureMode, measureRange, outputRangeValue:
        (!_measPinListIsNullOrEmpty) ? _outputRangeValue : null);
        TheLib.Execute.Wait(waitTime);
        _measPins = TheLib.Acquire.Dc.Measure(_pinsMeasure, sampleSize);
    }

    if (ShouldRunPostBody) {
        TheLib.Setup.Dc.Disconnect(_pinsAll);
        if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pinsAll);
        TheLib.Datalog.TestParametric(_measPins, forceValue);
    }
}

```

PreconditionPattern(PinList, string, double, double, Pattern,
string, double, int, PinList, double, string)

Parametric measurement by running preconditioning pattern, setting up force Pins and then measuring all force or optionally different measure Pins.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void PreconditionPattern(PinList forcePinList, string forceMode, double forceValue,
double clampValue, Pattern preconditionPat, string measureWhat, double measureRange, int
sampleSize = 1, PinList measPinList = null, double waitTime = 0, string setup = "")
```

Parameters

forcePinList PinList

Comma separated list of pin or pin groups representing the DC setup and/or measurement.

forceMode [string](#)

Force mode for each pin or pin group.

forceValue [double](#)

Force voltage or current for all pins or pin groups.

clampValue [double](#)

Clamp voltage or current for all pina or pin groups.

preconditionPat Pattern

Pattern to run to precondition the device before the parametric test.

measureWhat [string](#)

Measure either voltage or current for all measure pins or pin groups.

measureRange [double](#)

Expected voltage or current for all pins or pin groups to set the range.

sampleSize [int](#)

Optional. Number of samples to average for all pins or pin groups.

measPinList PinList

Optional. Comma separated list of measurement pins or pin groups, if different from forcePinList.

`waitForSetup` [double](#)

Optional. Settling time used after pin setup.

`setup` [string](#)

Optional. Setup setting to preconfigure the dib or device.

Details

Test Technique

- to be added

Implementation

The **Validation** section creates the `Pins` and `PatternInfo` objects, determines force & measure modes and confirms valid combinations.

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`. The preconditioning pattern is executed. For all pins specified in the `pinList` it disconnects any pin electronics, connects the dc path and turns on the gate.

The **Body** section applies the force condition on all pins, sets the measurement mode and performs the measurement on all pins in parallel after the specified `waitForSetup`.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Code Reference

```

[TestMethod, Steppable, CustomValidation]
public void PreconditionPattern(PinList forcePinList, string forceMode, double forceValue,
double clampValue, Pattern preconditionPat,
    string measureWhat, double measureRange, int sampleSize = 1, PinList measPinList = null,
double waitTime = 0.0, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(forcePinList, nameof(forcePinList), out _pinsForce);
        TheLib.Validate.Pins(forcePinList, nameof(forcePinList), out _pinsAll);
        _patternIsValid = !string.IsNullOrEmpty(preconditionPat);
        _measPinListIsNullOrEmpty = string.IsNullOrEmpty(measPinList);
        if (_patternIsValid) {
            TheLib.Validate.Pattern(preconditionPat, nameof(preconditionPat), out _pattern);
        }
        if (_measPinListIsNullOrEmpty) {
            _pinsMeasure = new Pins(forcePinList);
        } else {
            TheLib.Validate.Pins(measPinList, nameof(measPinList), out _pinsMeasure);
            _pinsAll.Add(measPinList);
        }
        TheLib.Validate.Enum(forceMode, nameof(forceMode), out _outputMode);
        TheLib.Validate.Enum(measureWhat, nameof(measureWhat), out _measureMode);
        _outputRangeValue = _outputMode == TLibOutputMode.ForceVoltage ? clampValue
: forceValue;
        _containsDigitalPins = _pinsAll.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_patternIsValid) TheLib.Execute.Digital.RunPattern(_pattern);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pinsAll);
        TheLib.Setup.Dc.Connect(_pinsAll);
    }

    if (ShouldRunBody) {
        TheLib.Setup.Dc.Force(_pinsForce, _outputMode, forceValue, clampValue);
        TheLib.Setup.Dc.SetMeter(_pinsMeasure, _measureMode, measureRange, outputRangeValue:
(!_measPinListIsNullOrEmpty) ? _outputRangeValue
        : null);
        TheLib.Execute.Wait(waitTime);
        _measPins = TheLib.Acquire.Dc.Measure(_pinsMeasure, sampleSize);
    }

    if (ShouldRunPostBody) {
        TheLib.Setup.Dc.Disconnect(_pinsAll);
    }
}

```

```
    if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pinsAll);
    TheLib.Datalog.TestParametric(_measPins, forceValue);
}
}
```

Namespace Demo_CSRA.Resistance

Overview

TestClass for all resistance related TestMethods.

Platform Specifics

Uses per test-instance test class object persistence (`[TestClass(Creation.TestInstance)]` attribute).

Classes

[Contact](#)

[RdsOn](#)

Class Contact

Namespace: [Demo_CSRA.Resistance](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class Contact : TestCodeBase
```

Inheritance

[object](#) ← TestCodeBase ← Contact

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
TestCodeBase.TheHdw , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
TestCodeBase.ShouldRunPreBody , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Baseline(PinList, string, double, double, double, double, double, double, string)

Performs a resistance measurement by forcing two values of voltage or current and measuring two currents or voltages.

```
[TestMethod]
[Steppable]
[CustomValidation]
```

```
public void Baseline(PinList forcePin, string forceMode, double forceFirstValue, double forceSecondValue, double clampValueOfForcePin, double measureFirstRange, double measureSecondRange, double waitTime = 0, string setup = "")
```

Parameters

forcePin [PinList](#)

Pin to force and measure.

forceMode [string](#)

The mode for forcing (e.g., Voltage or Current).

forceFirstValue [double](#)

First value to force.

forceSecondValue [double](#)

Second value to force.

clampValueOfForcePin [double](#)

Clamp Value of the force pin. May also set its range.

measureFirstRange [double](#)

The range for first measurement.

measureSecondRange [double](#)

The range for second measurement.

waitTime [double](#)

Optional. The wait time after forcing.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `config`. For all pins specified in the `forcePin` it disconnects any pin electronics, connects the dc path and turns on the gate.

The **Body** section applies a force current or voltage depending on the `forceMode`, sets the meter block and performs a voltage or current measurement on all pins specified in the `forcePin` in parallel after the specified `waitTime`. Settings are applied twice serially to achieve the delta between measurements. Finally, the resistance is calculated according to the forced values and the measured values.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- any dc paths from pins in `forcePin` are disconnected

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(PinList forcePin, string forceMode, double forceFirstValue, double
forceSecondValue, double clampValueOffForcePin,
    double measureFirstRange, double measureSecondRange, double waitTime = 0.0, string setup
= "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(forcePin, nameof(forcePin), out _pinsFirst);
        TheLib.Validate.Enum(forceMode.ToLower(), nameof(forceMode), out _outputMode);
        _measureMode = _outputMode == TLibOutputMode.ForceVoltage ? Measure.Current
        : Measure.Voltage;
        _forceFirst = new PinSite<double>(forcePin, forceFirstValue);
        _forceSecond = new PinSite<double>(forcePin, forceSecondValue);
        _containsDigitalPins = _pinsFirst.ContainsFeature(InstrumentFeature.Digital);
    }
}
```

```

if (ShouldRunPreBody) {
    TheLib.Setup.LevelsAndTiming.Apply(true);
    Services.Setup.Apply(setup);
    if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pinsFirst);
    TheLib.Setup.Dc.Connect(_pinsFirst);
}

if (ShouldRunBody) {
    TheLib.Setup.Dc.SetForceAndMeter(_pinsFirst, _outputMode, forceFirstValue,
forceFirstValue, clampValueOfForcePin, _measureMode,
    measureFirstRange);
    TheLib.Execute.Wait(waitTime);
    _measFirst = TheLib.Acquire.Dc.Measure(_pinsFirst);
    TheLib.Setup.Dc.SetForceAndMeter(_pinsFirst, _outputMode, forceSecondValue,
forceSecondValue, clampValueOfForcePin, _measureMode,
    measureSecondRange, false);
    TheLib.Execute.Wait(waitTime);
    _measSecond = TheLib.Acquire.Dc.Measure(_pinsFirst);
    _resistanceValue = (_outputMode == TLibOutputMode.ForceVoltage) ? (_forceFirst -
_forceSecond) / (_measFirst - _measSecond) :
    (_measFirst - _measSecond) / (_forceFirst - _forceSecond);
}

if (ShouldRunPostBody) {
    TheLib.Setup.Dc.Disconnect(_pinsFirst);
    if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pinsFirst);
    TheLib.Datalog.TestParametric(_resistanceValue);
}
}

```

Class RdsOn

Namespace: [Demo_CSRA.Resistance](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class RdsOn : TestCodeBase
```

Inheritance

[object](#) ← TestCodeBase ← RdsOn

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
TestCodeBase.TheHdw , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
TestCodeBase.ShouldRunPreBody , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Baseline(PinList, string, double, double, double, string, string)

Performs a resistance measurement by forcing voltage or current and measuring current or voltage on the same pin.

```
[TestMethod]
[Steppable]
[CustomValidation]
```

```
public void Baseline(PinList forcePin, string forceMode, double forceValue, double measureRange, double waitTime = 0, string labelOfStoredVoltage = "", string setup = "")
```

Parameters

forcePin PinList

Pin to force and measure.

forceMode [string](#)

The mode for forcing (e.g., Voltage or Current).

forceValue [double](#)

The value to force.

measureRange [double](#)

The range for the measurement.

waitTime [double](#)

Optional. The wait time after forcing.

labelOfStoredVoltage [string](#)

Optional. Label of a reference voltage from a previously stored measurement.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified **config**. For all pins specified in the **forcePin** it disconnects any pin electronics, connects the dc path and turns on the gate.

The **Body** section applies a force current or voltage depending on the **forceMode**, sets the meter block and performs a voltage or current measurement on all pins specified in the **forcePin** in parallel after the

specified `waitTime`. Finally, the resistance is calculated according to the forced and measured value. Optionally, resistance value can be calculated using the label of a reference voltage from a previously stored measurement `labelOfStoredVoltage`.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- any dc paths from pins in `forcePin` are disconnected

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(PinList forcePin, string forceMode, double forceValue, double
measureRange, double waitTime = 0,
    string labelOfStoredVoltage = "", string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(forcePin, nameof(forcePin), out _pinsFirst);
        if (labelOfStoredVoltage != "") _labelVoltage = new PinSite<double>(forcePin,
double.Parse(labelOfStoredVoltage));
        else _labelVoltage = new PinSite<double>(forcePin, 0.0);
        TheLib.Validate.Enum(forceMode.ToLower(), nameof(forceMode), out _outputMode);
        _measureMode = _outputMode == TLibOutputMode.ForceVoltage ? Measure.Current
: Measure.Voltage;
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        _forceFirst = new PinSite<double>(forcePin, forceValue);
        _containsDigitalPins = _pinsFirst.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pinsFirst);
        TheLib.Setup.Dc.Connect(_pinsFirst);
    }
}
```

```

    }

    if (ShouldRunBody) {
        TheLib.Setup.Dc.SetForceAndMeter(_pinsFirst, _outputMode, forceValue, forceValue,
measureRange, _measureMode, measureRange);
        TheLib.Execute.Wait(waitTime);
        _measFirst = TheLib.Acquire.Dc.Measure(_pinsFirst);
        _resistanceValue = (_outputMode == TLibOutputMode.ForceVoltage) ? (_forceFirst -
_labelVoltage) / _measFirst :
            (_measFirst - _labelVoltage) / _forceFirst;
    }

    if (ShouldRunPostBody) {
        TheLib.Setup.Dc.Disconnect(_pinsFirst);
        if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pinsFirst);
        TheLib.Datalog.TestParametric(_resistanceValue);
    }
}

```

FourPinsTwoForceTwoMeasure(PinList, double, double, PinList, double, double, PinList, PinList, double, double, double, string)

Performs a resistance delta measurement by forcing a current on first pin, voltage on second pin and measuring a delta voltage on two other pins.

```

[TestMethod]
[Steppable]
[CustomValidation]
public void FourPinsTwoForceTwoMeasure(PinList forceFirstPin, double forceValueFirstPin,
double clampValueOfForceFirstPin, PinList forceSecondPin, double forceValueSecondPin, double
clampValueOfForceSecondPin, PinList measureFirstPin, PinList measureSecondPin, double
measureRangeFirstPin, double measureRangeSecondPin, double waitTime = 0, string setup = "")

```

Parameters

forceFirstPin PinList

First pin to force Current.

forceValueFirstPin [double](#)

Value to force on first pin.

`clampValueOfForceFirstPin` [double](#)

Clamp Value of the first force pin. May also set its range.

`forceSecondPin` [PinList](#)

Second pin to force Voltage.

`forceValueSecondPin` [double](#)

Value to force on second pin.

`clampValueOfForceSecondPin` [double](#)

Clamp Value of the second force pin. May also set its range.

`measureFirstPin` [PinList](#)

First pin to measure Voltage.

`measureSecondPin` [PinList](#)

Second pin to measure Voltage.

`measureRangeFirstPin` [double](#)

Measure range of first measure pins.

`measureRangeSecondPin` [double](#)

Measure range of second measure pins.

`waitTime` [double](#)

Optional. Wait time after forcing.

`setup` [string](#)

Optional. Name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `config`. For all pins specified in `forceFirstPin`, `forceSecondPin`, `measureFirstPin` and `measureSecondPin` it disconnects any pin electronics, connects the dc path and turns on the gate.

The **Body** section applies a force current on all pins specified in the `forceFirstPin` and force voltage on all pins specified in the `forceSecondPin`. Configures High Impedance mode, initializes the meter block, and performs a voltage measurement across all pins in parallel specified in `measureFirstPin` and in `measureSecondPin` after the specified `waitTime`. Finally, the resistance is calculated according to the forced current value and the measured values.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- any dc paths from pins in `forceFirstPin`, `forceSecondPin`, `measureFirstPin` and `measureSecondPin` are disconnected

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void FourPinsTwoForceTwoMeasure(PinList forceFirstPin, double forceValueFirstPin,
double clampValueOfForceFirstPin, PinList forceSecondPin,
double forceValueSecondPin, double clampValueOfForceSecondPin, PinList measureFirstPin,
PinList measureSecondPin, double measureRangeFirstPin,
double measureRangeSecondPin, double waitTime = 0, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(forceFirstPin, nameof(forceFirstPin), out _pinsFirst);
        TheLib.Validate.Pins(forceSecondPin, nameof(forceSecondPin), out _pinsSecond);
        TheLib.Validate.Pins(measureFirstPin, nameof(measureFirstPin), out _pinsFirstMeas);
        TheLib.Validate.Pins(measureSecondPin, nameof(measureSecondPin),
out _pinsSecondMeas);
        _allPins = new Pins(string.Join(", ", forceFirstPin, forceSecondPin,
measureFirstPin, measureSecondPin));
    }
}
```

```

        _allMeasPins = new Pins(string.Join(", ", measureFirstPin, measureSecondPin));
        _forceFirst = new PinSite<double>(forceFirstPin, forceValueFirstPin);
        _containsDigitalPins = _allPins.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_allPins);
        TheLib.Setup.Dc.Connect(_allPins);
    }

    if (ShouldRunBody) {
        TheLib.Setup.Dc.ForceHiZ(_allMeasPins);
        TheLib.Setup.Dc.Force(_pinsFirst, TLibOutputMode.ForceCurrent, forceValueFirstPin,
forceValueFirstPin, clampValueOfForceFirstPin);
        TheLib.Setup.Dc.Force(_pinsSecond, TLibOutputMode.ForceVoltage, forceValueSecondPin,
forceValueSecondPin, clampValueOfForceSecondPin);
        TheLib.Setup.Dc.SetMeter(_pinsFirstMeas, Measure.Voltage, measureRangeFirstPin);
        TheLib.Setup.Dc.SetMeter(_pinsSecondMeas, Measure.Voltage, measureRangeSecondPin);
        TheLib.Execute.Wait(waitTime);
        _measFirst = TheLib.Acquire.Dc.Measure(_pinsFirstMeas);
        _measSecond = TheLib.Acquire.Dc.Measure(_pinsSecondMeas);
        _resistanceValue = (_measFirst - _measSecond) / _forceFirst;
    }

    if (ShouldRunPostBody) {
        TheLib.Setup.Dc.Disconnect(_allPins);
        if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_allPins);
        TheLib.Datalog.TestParametric(_resistanceValue);
    }
}

```

ThreePinsOneForceTwoMeasure(PinList, double, double, PinList, double, PinList, double, string)

Performs a resistance measurement by forcing a current on one pin and measuring on two other pins.

```

[TestMethod]
[Steppable]
[CustomValidation]
public void ThreePinsOneForceTwoMeasure(PinList forcePin, double forceCurrentPin, double

```

```
clampValueOfForcePin, PinList measureFirstPin, double measureRangeFirstPin, PinList  
measureSecondPin, double measureRangeSecondPin, double waitTime = 0, string setup = "")
```

Parameters

forcePin PinList

Pin to force.

forceCurrentPin [double](#)

Current to force.

clampValueOfForcePin [double](#)

Clamp Value of the force pin. May also set its range.

measureFirstPin PinList

First pin to measure voltage.

measureRangeFirstPin [double](#)

Measure range of first measure pin.

measureSecondPin PinList

Second pin to measure voltage.

measureRangeSecondPin [double](#)

Measure range of second measure pin.

waitTime [double](#)

Optional. Wait time after forcing.

setup [string](#)

Optional. Name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `config`. For all pins specified in the `forcePin`, `measureFirstPin` and `measureSecondPin` it disconnects any pin electronics, connects the dc path and turns on the gate.

The **Body** section applies a force current on all pins specified in the `forcePin`. Configures High Impedance mode, initializes the meter block, and performs a voltage measurement on all pins in parallel specified in `measureFirstPin` and in `measureSecondPin` after the specified `waitTime`. Finally, the resistance is calculated according to the forced value and the measured values.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- any dc paths from pins in `forcePin`, `measureFirstPin` and `measureSecondPin` are disconnected

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void ThreePinsOneForceTwoMeasure(PinList forcePin, double forceCurrentPin, double
clampValueOfForcePin, PinList measureFirstPin,
double measureRangeFirstPin, PinList measureSecondPin, double measureRangeSecondPin,
double waitTime = 0, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(forcePin, nameof(forcePin), out _pinsFirst);
        TheLib.Validate.Pins(measureFirstPin, nameof(measureFirstPin), out _pinsFirstMeas);
        TheLib.Validate.Pins(measureSecondPin, nameof(measureSecondPin),
out _pinsSecondMeas);
        _forceFirst = new PinSite<double>(forcePin, forceCurrentPin);
        _allPins = new Pins(string.Join(", ", forcePin, measureFirstPin, measureSecondPin));
        _allMeasPins = new Pins(string.Join(", ", measureFirstPin, measureSecondPin));
        _containsDigitalPins = _allPins.ContainsFeature(InstrumentFeature.Digital);
    }
}
```

```

if (ShouldRunPreBody) {
    TheLib.Setup.LevelsAndTiming.Apply(true);
    Services.Setup.Apply(setup);
    if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_allPins);
    TheLib.Setup.Dc.Connect(_allPins);
}

if (ShouldRunBody) {
    TheLib.Setup.Dc.ForceHiZ(_allMeasPins);
    TheLib.Setup.Dc.Force(_pinsFirst, TLibOutputMode.ForceCurrent, forceCurrentPin,
forceCurrentPin, clampValueOfForcePin);
    TheLib.Setup.Dc.SetMeter(_pinsFirstMeas, Measure.Voltage, measureRangeFirstPin);
    TheLib.Setup.Dc.SetMeter(_pinsSecondMeas, Measure.Voltage, measureRangeSecondPin);
    TheLib.Execute.Wait(waitTime);
    _measFirst = TheLib.Acquire.Dc.Measure(_pinsFirstMeas);
    _measSecond = TheLib.Acquire.Dc.Measure(_pinsSecondMeas);
    _resistanceValue = (_measFirst - _measSecond) / _forceFirst;
}

if (ShouldRunPostBody) {
    TheLib.Setup.Dc.Disconnect(_allPins);
    if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_allPins);
    TheLib.Datalog.TestParametric(_resistanceValue);
}
}

```

TwoPinsDeltaForceDeltaMeasure(PinList, string, double, double, double, PinList, double, double, double, string)

Performs a resistance delta measurement by forcing two force values on one pin and measuring with a second pin.

```

[TestMethod]
[Steppable]
[CustomValidation]
public void TwoPinsDeltaForceDeltaMeasure(PinList forcePin, string forceMode, double
forceFirstValue, double forceSecondValue, double clampValueOfForcePin, PinList measurePin,
double measureFirstRange, double measureSecondRange, double waitTime = 0, string setup = "")

```

Parameters

forcePin PinList

Pin to force.

forceMode [string](#)

Force Mode of force pin (e.g., Voltage or Current).

forceFirstValue [double](#)

First value to force.

forceSecondValue [double](#)

Second value to force to calculate a delta.

clampValueOfForcePin [double](#)

Clamp Value of the force pin. May also set its range.

measurePin PinList

Pin to measure.

measureFirstRange [double](#)

First range for the measurement.

measureSecondRange [double](#)

Second range for the measurement.

waitTime [double](#)

Optional. Wait time after forcing.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `config`. For all pins specified in the `forcePin` and `measurePin` it disconnects any pin electronics, connects the dc path and turns on the gate.

The **Body** section applies a force current or voltage depending on the `forceMode`. Configures High Impedance mode, initializes the meter block, and performs a voltage or current measurement on all pins in parallel specified in the `measurePin` after the specified `waitTime`. Settings are applied twice serially to achieve the delta between measurements. Finally, the resistance is calculated according to the forced values and the measured values.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- any dc paths from pins in `forcePin` and `measurePin` are disconnected

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void TwoPinsDeltaForceDeltaMeasure(PinList forcePin, string forceMode, double
forceFirstValue, double forceSecondValue,
    double clampValueOfForcePin, PinList measurePin, double measureFirstRange, double
measureSecondRange, double waitTime = 0, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(forcePin, nameof(forcePin), out _pinsFirst);
        TheLib.Validate.Pins(measurePin, nameof(measurePin), out _pinsFirstMeas);
        TheLib.Validate.Enum(forceMode.ToLower(), nameof(forceMode), out _outputMode);
        _measureMode = _outputMode == TLibOutputMode.ForceVoltage ? Measure.Current
: Measure.Voltage;
        _forceFirst = new PinSite<double>(forcePin, forceFirstValue);
        _forceSecond = new PinSite<double>(forcePin, forceSecondValue);
        _allPins = new Pins(forcePin);
        _allPins.Add(measurePin);
        _containsDigitalPins = _allPins.ContainsFeature(InstrumentFeature.Digital);
```

```

}

if (ShouldRunPreBody) {
    TheLib.Setup.LevelsAndTiming.Apply(true);
    Services.Setup.Apply(setup);
    if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_allPins);
    TheLib.Setup.Dc.Connect(_allPins);
}

if (ShouldRunBody) {
    TheLib.Setup.Dc.ForceHiZ(_pinsFirstMeas);
    TheLib.Setup.Dc.Force(_pinsFirst, _outputMode, forceFirstValue,
forceFirstValue, clampValueOfForcePin);
    TheLib.Setup.Dc.SetMeter(_pinsFirstMeas, _measureMode, measureFirstRange);
    TheLib.Execute.Wait(waitTime);
    _measFirst = TheLib.Acquire.Dc.Measure(_pinsFirstMeas);

    TheLib.Setup.Dc.Force(_pinsFirst, _outputMode, forceSecondValue,
forceSecondValue, clampValueOfForcePin);
    TheLib.Setup.Dc.SetMeter(_pinsFirstMeas, _measureMode, measureSecondRange);
    TheLib.Execute.Wait(waitTime);
    _measSecond = TheLib.Acquire.Dc.Measure(_pinsFirstMeas);

    _resistanceValue = (_outputMode == TLibOutputMode.ForceVoltage) ? (_forceFirst -
_forceSecond) / (_measFirst - _measSecond) :
    (_measFirst - _measSecond) / (_forceFirst - _forceSecond);
}

if (ShouldRunPostBody) {
    TheLib.Setup.Dc.Disconnect(_allPins);
    if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_allPins);
    TheLib.Datalog.TestParametric(_resistanceValue);
}
}

```

TwoPinsOneForceOneMeasure(PinList, string, double, double, PinList, double, double, string, string)

Performs a resistance measurement by forcing voltage or current on one pin and measuring current or voltage on second pin.

[TestMethod]
[Steppable]

```
[CustomValidation]
public void TwoPinsOneForceOneMeasure(PinList forcePin, string forceMode, double forceValue,
double clampValueOfForcePin, PinList measurePin, double measureRange, double waitTime = 0,
string labelOfStoredVoltage = "", string setup = "")
```

Parameters

forcePin PinList

Pin to force.

forceMode [string](#)

The mode for forcing (e.g., Voltage or Current).

forceValue [double](#)

The value to force.

clampValueOfForcePin [double](#)

Clamp Value of the force pin. May also set its range.

measurePin PinList

Pin to measure.

measureRange [double](#)

The range for measurement.

waitTime [double](#)

Optional. The wait time after forcing.

labelOfStoredVoltage [string](#)

Optional. Label of a reference voltage from a previously stored measurement.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `config`. For all pins specified in the `forcePin` and `measurePin` it disconnects any pin electronics, connects the dc path and turns on the gate..

The **Body** section applies a force current or voltage depending on the `forceMode` on all pins specified in the `forcePin`. Configures High Impedance mode, initializes the meter block, and performs a voltage or current measurement on all pins in parallel specified in the `measurePin` after the specified `waitTime`. Finally, resistance is calculated according to the forced and measured value. Optionally, resistance value can be calculated using the label of a reference voltage from a previously stored measurement `labelOfStoredVoltage`.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- any dc paths from pins in `forcePin` and `measurePin` are disconnected

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void TwoPinsOneForceOneMeasure(PinList forcePin, string forceMode, double forceValue,
double clampValueOfForcePin, PinList measurePin,
    double measureRange, double waitTime = 0, string labelOfStoredVoltage = "", string setup
= "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(forcePin, nameof(forcePin), out _pinsFirst);
        TheLib.Validate.Pins(measurePin, nameof(measurePin), out _pinsFirstMeas);
        _forceFirst = new PinSite<double>(forcePin, forceValue);
        if (labelOfStoredVoltage != "") _labelVoltage = new PinSite<double>(forcePin,
double.Parse(labelOfStoredVoltage));
    }
}
```

```

        else _labelVoltage = new PinSite<double>(forcePin, 0.0);
        TheLib.Validate.Enum(forceMode.ToLower(), nameof(forceMode), out _outputMode);
        _measureMode = _outputMode == TLibOutputMode.ForceVoltage ? Measure.Current
: Measure.Voltage;
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        _allPins = new Pins(forcePin);
        _allPins.Add(measurePin);
        _containsDigitalPins = _allPins.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_allPins);
        TheLib.Setup.Dc.Connect(_allPins);
    }

    if (ShouldRunBody) {
        TheLib.Setup.Dc.ForceHiZ(_pinsFirstMeas);
        TheLib.Setup.Dc.Force(_pinsFirst, _outputMode, forceValue,
forceValue, clampValueOfForcePin);
        TheLib.Setup.Dc.SetMeter(_pinsFirstMeas, _measureMode, measureRange);
        TheLib.Execute.Wait(waitTime);
        _measFirst = TheLib.Acquire.Dc.Measure(_pinsFirstMeas);
        _resistanceValue = (_outputMode == TLibOutputMode.ForceVoltage) ? (_forceFirst -
_labelVoltage) / _measFirst :
            (_measFirst - _labelVoltage) / _forceFirst;
    }

    if (ShouldRunPostBody) {
        TheLib.Setup.Dc.Disconnect(_allPins);
        if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_allPins);
        TheLib.Datalog.TestParametric(_resistanceValue);
    }
}

```

Namespace Demo_CSRA.ScanNetwork

Classes

[ScanNetworkPattern](#)

Class ScanNetworkPattern

Namespace: [Demo_CSRA.ScanNetwork](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class ScanNetworkPattern : TestCodeBase
```

Inheritance

[object](#) ← TestCodeBase ← ScanNetworkPattern

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
TestCodeBase.TheHdw , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
TestCodeBase.ShouldRunPreBody , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Baseline(Pattern, string, string, string, bool, bool)

Executes a ScanNetwork pattern(set) and retrieve per core/icl instance results for each site.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Baseline(Pattern scanNetworkPattern, string setupPatternCsv = "", string
```

```
endPatternCsv = "", string setup = "", bool runDiagnosis = false, bool multiCoreDiagnosis = false)
```

Parameters

scanNetworkPattern Pattern

The ScanNetwork pattern(set) to be executed.

setupPatternCsv [string](#)

Optional. The setup.csv file that comes with the ScanNetwork_setup pattern or the concatenated ScanNetwork pattern (stil)

endPatternCsv [string](#)

Optional. The end.csv file that comes with the ScanNetwork_end pattern. Leave blank if the pattern(stil) is a concatenated ScanNetwork pattern.

setup [string](#)

Optional. Setup to be applied before the pattern runs.

runDiagnosis [bool](#)

Optional. Whether to perform Diagnosis on failed cores after pattern execution.

multiCoreDiagnosis [bool](#)

Optional. Whether to enable multiple core instances during Diagnosis reburst.

false: Diagnosis will be performed one core instance at a time;

true: Diagnosis will be performed on as many cores as possible to minimize the number of reburst.

Namespace Demo_CSRA.Search

Overview

TestClass for all search related TestMethods.

Platform Specifics

Uses per test-instance test class object persistence (`[TestClass(Creation.TestInstance)]` attribute).

Classes

[Functional](#)

[Parametric](#)

Class Functional

Namespace: [Demo_CSRA.Search](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class Functional : TestCodeBase
```

Inheritance

[object](#) ← TestCodeBase ← Functional

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
TestCodeBase.TheHdw , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
TestCodeBase.ShouldRunPreBody , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Binary(Pattern, PinList, double, double, double, bool, double, string)

The search range is divided in half at each iteration, with a check performed on the midpoint, and the search stops once the target condition is met. The input value where the condition passes is then returned as the result.

```
[TestMethod]
[Steppable]
```

```
[CustomValidation]
public void Binary(Pattern pattern, PinList forcePins, double from, double to, double
minDelta, bool invertedOutput, double waitTime, string setup = "")
```

Parameters

pattern Pattern

The pattern to run.

forcePins PinList

The pins that are being forced. The support pin types can be DC(DCVI, DCSV and PPMU) and Digital(Vih level).

from [double](#)

The starting value of the linear input ramp.

to [double](#)

The stopping value of the linear input ramp.

minDelta [double](#)

The minimum allowable difference between successive input values, used to determine when the search should stop.

invertedOutput [bool](#)

A flag indicating whether the output is inverted, affecting the search logic.

waitTime [double](#)

The wait time per step during ramp execution, used to delay measurement after each force transition.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`. For the force pin, specified in `forcePins`, it disconnects any pin electronics and connects the DC path.

The **Body** section performs a binary search (via `Vih` for the Digital pin, voltage for DCVI/DCVS/PPMU pin) until it determines the device input condition for which the pattern passes. Finally, it returns the input value that produces a passing pattern result. If no such value is found, the function returns `-999` as a failure indicator.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, logs a parametric datalog.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

Pin features supporting DCVI, DCVS, PPMU, and Digital, note that PPMU must be used in combination with either DCVI or DCVS to function.

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Binary(Pattern pattern, PinList forcePins, double from, double to, double
minDelta, bool invertedOutput, double waitTime,
    string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pattern(pattern, nameof(pattern), out _pattern);
        TheLib.Validate.Pins(forcePins, nameof(forcePins), out _pins);
        TheLib.Validate.GreaterThan(minDelta, 0, nameof(minDelta));
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        _containsDcvicDcvs = _pins.ContainsFeature(InstrumentFeature.Dcvic) ||
        _pins.ContainsFeature(InstrumentFeature.Dcvs);
        if (_containsDcvicDcvs) _containsDigitalPins =
        _pins.ContainsFeature(InstrumentFeature.Digital);
    }
}
```

```

if (ShouldRunPreBody) {
    TheLib.Setup.LevelsAndTiming.Apply(true);
    Services.Setup.Apply(setup);
    if (_containsDcvicv) {
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
        TheLib.Setup.Dc.Connect(_pins);
    }
}

if (ShouldRunBody) {
    if (_containsDcvicv) {
        TheLib.Setup.Dc.ForceV(_pins, (from + to) / 2);
        TheLib.Execute.Wait(waitTime); // first step may be bigger than the subsequent
ones, use 2x settling
    }
    _values = TheLib.Acquire.Search.BinarySearch(from, to, minDelta, invertedOutput,
(forceValue) => {
        ForEachSite(site => {
            if (_containsDcvicv) TheLib.Setup.Dc.Modify(_pins,
voltage: forceValue[site]);
            else TheLib.Setup.Digital.ModifyPinsLevels(pins: _pins, levelsType:
ChPinLevel.Vih, levelsValue: forceValue[site]);
        });
        TheLib.Execute.Wait(waitTime);
        TheLib.Execute.Digital.RunPattern(_pattern);
        return TheLib.Acquire.Digital.PatternResults();
    },
    patResult => patResult,
    _notFoundResult
);
}
}

if (ShouldRunPostBody) {
    if (_containsDcvicv) {
        TheLib.Setup.Dc.Disconnect(_pins);
        if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
    }
    TheLib.Datalog.TestParametric(_values);
}
}

```

LinearFull(Pattern, PinList, double, double, int, double, string)

The measurements across the entire range are traversed without being evaluated during the linear search, after which the device input condition for which the pattern passes is provided.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void LinearFull(Pattern pattern, PinList forcePins, double from, double to, int
count, double waitTime, string setup = "")
```

Parameters

pattern Pattern

The pattern to run.

forcePins PinList

The pins that are being forced. The support pin types can be DC(DCVI, DCSV and PPMU) and Digital(Vih level).

from [double](#)

The starting value of the linear input ramp.

to [double](#)

The stopping value of the linear input ramp.

count [int](#)

The number of input points for which the search is performed.

waitTime [double](#)

The wait time per step during ramp execution, used to delay measurement after each force transition.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`. For the force pin, specified in `forcePins`, it disconnects any pin electronics and connects the DC path.

The **Body** section performs an unconditional linear search (via `Vih` for the Digital pin, voltage for DCVI/DCVS/PPMU pin) and determines the device input condition for which the pattern passes. Finally, it returns the input value that results in a passing pattern result. If no such value is found, the function returns `-999` as a failure indicator.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, logs a parametric datalog.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

Pin features supporting DCVI, DCVS, PPMU, and Digital, note that PPMU must be used in combination with either DCVI or DCVS to function.

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void LinearFull(Pattern pattern, PinList forcePins, double from, double to, int
count, double waitTime, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pattern(pattern, nameof(pattern), out _pattern);
        TheLib.Validate.Pins(forcePins, nameof(forcePins), out _pins);
        TheLib.Validate.GreaterOrEqual(count, 2, nameof(count));
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        _containsDcviDcvs = _pins.ContainsFeature(InstrumentFeature.Dcvi) ||
        _pins.ContainsFeature(InstrumentFeature.Dcvs);
        if (_containsDcviDcvs) _containsDigitalPins =
        _pins.ContainsFeature(InstrumentFeature.Digital);
    }
}
```

```

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_containsDcvicDcvs) {
            if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
            TheLib.Setup.Dc.Connect(_pins);
        }
    }

    if (ShouldRunBody) {
        _measurements = [];
        if (_containsDcvicDcvs) {
            TheLib.Setup.Dc.ForceV(_pins, from);
            TheLib.Execute.Wait(waitTime); // first step may be bigger than the subsequent ones, use 2x settling
        }
        double increment = TheLib.Acquire.Search.LinearFullFromToCount(from, to, count,
        (forceValue) => {
            if (_containsDcvicDcvs) TheLib.Setup.Dc.Modify(_pins, voltage: forceValue);
            else TheLib.Setup.Digital.ModifyPinsLevels(pins: _pins, levelsType:
            ChPinLevel.Vih, levelsValue: forceValue);
            TheLib.Execute.Wait(waitTime);
            TheLib.Execute.Digital.RunPattern(_pattern);
            _measurements.Add(TheLib.Acquire.Digital.PatternResults());
        });
        _results = TheLib.Execute.Search.LinearFullProcess(_measurements, from, increment,
        0, _notFoundResult, condition => condition);
    }

    if (ShouldRunPostBody) {
        if (_containsDcvicDcvs) {
            TheLib.Setup.Dc.Disconnect(_pins);
            if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
        }
        TheLib.Datalog.TestParametric(_results);
    }
}

```

LinearStop(Pattern, PinList, double, double, int, double, string)

The measurements across the range are traversed with an evaluation performed at each iteration, and the search is stopped once the pattern passes (on all sites). The device input condition for which the pattern passes is then provided.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void LinearStop(Pattern pattern, PinList forcePins, double from, double to, int
count, double waitTime, string setup = "")
```

Parameters

pattern Pattern

The pattern to run.

forcePins PinList

The pins that are being forced. The support pin types can be DC(DCVI, DCSV and PPMU) and Digital(Vih level).

from [double](#)

The starting value of the linear input ramp.

to [double](#)

The stopping value of the linear input ramp.

count [int](#)

The number of input points for which the search is performed.

waitTime [double](#)

The wait time per step during ramp execution, used to delay measurement after each force transition.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`. For the force pin, specified in `forcePins`, it disconnects any pin electronics and connects the DC path.

The **Body** section performs a linear search (via `Vih` for the Digital pin, voltage for DCVI/DCVS/PPMU pin) until it determines the device input condition for which the pattern passes. Finally, it returns the input value that produces a passing pattern result. If no such value is found, the function returns `-999` as a failure indicator.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, logs a parametric datalog.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

Pin features supporting DCVI, DCVS, PPMU, and Digital, note that PPMU must be used in combination with either DCVI or DCVS to function.

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void LinearStop(Pattern pattern, PinList forcePins, double from, double to, int
count, double waitTime, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pattern(pattern, nameof(pattern), out _pattern);
        TheLib.Validate.Pins(forcePins, nameof(forcePins), out _pins);
        TheLib.Validate.GreaterOrEqual(count, 2, nameof(count));
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        _containsDcviDcvs = _pins.ContainsFeature(InstrumentFeature.Dcvi) ||
        _pins.ContainsFeature(InstrumentFeature.Dcvs);
        if (_containsDcviDcvs) _containsDigitalPins =
        _pins.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
    }
}
```

```

Services.Setup.Apply(setup);
if (_containsDcviDcvs) {
    if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
    TheLib.Setup.Dc.Connect(_pins);
}
}

if (ShouldRunBody) {
    if (_containsDcviDcvs) {
        TheLib.Setup.Dc.ForceV(_pins, from);
        TheLib.Execute.Wait(waitTime); // first step may be bigger than the subsequent
ones, use 2x settling
    }
    _values = TheLib.Acquire.Search.LinearStopFromToCount(from, to, count, 0,
_notFoundResult, (forceValue) => {
        if (_containsDcviDcvs) TheLib.Setup.Dc.Modify(_pins, voltage: forceValue);
        else TheLib.Setup.Digital.ModifyPinsLevels(pins: _pins, levelsType:
ChPinLevel.Vih, levelsValue: forceValue);
        TheLib.Execute.Wait(waitTime);
        TheLib.Execute.Digital.RunPattern(_pattern);
        return TheLib.Acquire.Digital.PatternResults();
},
patResult => patResult
);
}
}

if (ShouldRunPostBody) {
    if (_containsDcviDcvs) {
        TheLib.Setup.Dc.Disconnect(_pins);
        if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
    }
    TheLib.Datalog.TestParametric(_values);
}
}

```

Class Parametric

Namespace: [Demo_CSRA.Search](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class Parametric : TestCodeBase
```

Inheritance

[object](#) ← TestCodeBase ← Parametric

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
TestCodeBase.TheHdw , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
TestCodeBase.ShouldRunPreBody , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

LinearFull(PinList, PinList, double, double, int, double, double, double, string)

Voltage values across the entire range are traversed without being evaluated during the linear search. Subsequently, the device input condition for which the output pin voltage exceeds the threshold is determined.

```
[TestMethod]
[Steppable]
```

```
[CustomValidation]
public void LinearFull(PinList forcePins, PinList measurePin, double from, double to, int
count, double threshold, double clampCurrent, double waitTime, string setup = "")
```

Parameters

forcePins PinList

The pins that are being forced.

measurePin PinList

The pin that is being measured. The measurement is performed for a single pin.

from [double](#)

The starting value of the linear input ramp.

to [double](#)

The stopping value of the linear input ramp.

count [int](#)

The number of input points for which the search is performed.

threshold [double](#)

The value for which the output meets the required condition for the searched input value.

clampCurrent [double](#)

The value to clamp for force pin.

waitTime [double](#)

The wait time per step during ramp execution, used to delay measurement after each force transition.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`. For the force pin and the measure pin, specified in `forcePins` and `measurePin`, it disconnects any pin electronics and connects the DC path.

The **Body** section applies a voltage force of the `start` value and turns on the gate for `forcePins`, sets the measurement block for `measurePin`, and performs an unconditional linear search up to the `stop` value with a step count of `step`. Finally, it returns the input value that results in an output closest to the voltage `threshold` point.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- any dc paths from pins in `forcePin` are disconnected

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void LinearFull(PinList forcePins, PinList measurePin, double from, double to, int
count, double threshold, double clampCurrent,
double waitTime, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(forcePins, nameof(forcePins), out _forcePin);
        TheLib.Validate.Pins(measurePin, nameof(measurePin), out _measurePin);
        TheLib.Validate.GreaterOrEqual(count, 2, nameof(count));
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        if (_measurePin.Count() != 1) Services.Alert.Error($"Only one pin can be used for
measurement. The number of measurement pins is invalid.");
        _allPins = new Pins($"{forcePins.Value}, {_measurePin.Value}");
        _containsDigitalPins = _allPins.ContainsFeature(InstrumentFeature.Digital);
        _voltageRange = (from > to) ? from : to;
    }
}
```

```

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_allPins);
        TheLib.Setup.Dc.Connect(_allPins);
    }

    if (ShouldRunBody) {
        _measuredDcValues = [];
        TheLib.Setup.Dc.ForceHiZ(_measurePin);
        TheLib.Setup.Dc.ForceV(_forcePin, from, clampCurrent, _voltageRange,
clampCurrent, true);
        TheLib.Setup.Dc.SetMeter(_measurePin, Measure.Voltage, _voltageRange);
        double increment = TheLib.Acquire.Search.LinearFullFromToCount(from, to, count,
(forceValue) => {
            TheLib.Setup.Dc.ForceV(_forcePin, forceValue, gateOn: false);
            TheLib.Execute.Wait(waitTime);
            _measuredDcValues.Add(TheLib.Acquire.Dc.Measure(_measurePin).First());
        });
        _results = TheLib.Execute.Search.LinearFullProcess(_measuredDcValues, from,
increment, 0, _notFoundResult, (measuredValue) => threshold < measuredValue);
    }

    if (ShouldRunPostBody) {
        TheLib.Setup.Dc.Disconnect(_allPins);
        if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_allPins);
        TheLib.Datalog.TestParametric(_results);
    }
}

```

LinearStop(PinList, PinList, double, double, int, double, double, double, string)

Voltage values across the range are traversed with an evaluation performed at each iteration, and the search is terminated once the objective has been reached across all sites. The device input condition for which the output pin voltage exceeds the threshold is then determined.

```

[TestMethod]
[Steppable]
[CustomValidation]

```

```
public void LinearStop(PinList forcePins, PinList measurePin, double from, double to, int count, double threshold, double clampCurrent, double waitTime, string setup = "")
```

Parameters

forcePins PinList

The pins that are being forced.

measurePin PinList

The pin that is being measured. The measurement is performed for a single pin.

from [double](#)

The starting value of the linear input ramp.

to [double](#)

The stopping value of the linear input ramp.

count [int](#)

The number of input points for which the search is performed.

threshold [double](#)

The value for which the output meets the required condition for the searched input value.

clampCurrent [double](#)

The value to clamp for force pin.

waitTime [double](#)

The wait time per step during ramp execution, used to delay measurement after each force transition.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`. For the force pin and the measure pin, specified in `forcePins` and `measurePin`, it disconnects any pin electronics and connects the DC path.

The **Body** section applies a voltage force of the `start` value and turns on the gate for `forcePins`, sets the measurement block for `measurePin`, and initiates a search until the output value reaches the voltage target. It stops when all sites have reached that condition, and returns the first input value that results in an output beyond the `threshold` point. If no such value is found, the function returns `-999` as a failure indicator.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- any dc paths from pins in `forcePin` are disconnected

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void LinearStop(PinList forcePins, PinList measurePin, double from, double to, int
count, double threshold, double clampCurrent,
double waitTime, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(forcePins, nameof(forcePins), out _forcePin);
        TheLib.Validate.Pins(measurePin, nameof(measurePin), out _measurePin);
        TheLib.Validate.GreaterOrEqual(count, 2, nameof(count));
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        if (_measurePin.Count() != 1) Services.Alert.Error($"Only one pin can be used for
measurement. The number of measurement pins is invalid.");
        _allPins = new Pins($"{forcePins.Value}, {_measurePin.Value}");
        _containsDigitalPins = _allPins.ContainsFeature(InstrumentFeature.Digital);
        _voltageRange = (from > to) ? from : to;
    }
}
```

```

}

if (ShouldRunPreBody) {
    TheLib.Setup.LevelsAndTiming.Apply(true);
    Services.Setup.Apply(setup);
    if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_allPins);
    TheLib.Setup.Dc.Connect(_allPins);
}

if (ShouldRunBody) {
    TheLib.Setup.Dc.ForceHiZ(_measurePin);
    TheLib.Setup.Dc.ForceV(_forcePin, from, clampCurrent, _voltageRange,
clampCurrent, true);
    TheLib.Setup.Dc.SetMeter(_measurePin, Measure.Voltage, _voltageRange);
    _results = TheLib.Acquire.Search.LinearStopFromToCount(from, to, count, 0,
_notFoundResult, (forceValue) => {
        TheLib.Setup.Dc.ForceV(_forcePin, forceValue, gateOn: false);
        TheLib.Execute.Wait(waitTime);
        return TheLib.Acquire.Dc.Measure(_measurePin).First();
    }, (measuredValue) => threshold < measuredValue);
}

if (ShouldRunPostBody) {
    TheLib.Setup.Dc.Disconnect(_allPins);
    if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_allPins);
    TheLib.Datalog.TestParametric(_results);
}
}

```

Namespace Demo_CSRA.SupplyCurrent

Classes

[Dynamic](#)

[Static](#)

Class Dynamic

Namespace: [Demo_CSRA.SupplyCurrent](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class Dynamic : TestCodeBase
```

Overview

TestClass for all dynamic supply current related TestMethods.

Platform Specifics

Uses per test-instance test class object persistence ([\[TestClass\(Creation.TestInstance\)\]](#) attribute).

Inheritance

[object](#) ← TestCodeBase ← Dynamic

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
[TestCodeBase.TheHdw](#) , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
[TestCodeBase.ShouldRunPreBody](#) , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

[Baseline\(PinList, double, double, double, double, Pattern, int, string\)](#)

Performs a predefined number of current measurements synchronized by a Flag stop with the pattern.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Baseline(PinList pinList, double forceValue, double measureRange, double
clampValue, double waitTime, Pattern pattern, int stops, string setup = "")
```

Parameters

pinList PinList

List of pin or pin group names.

forceValue [double](#)

The value to force.

measureRange [double](#)

The range for measurement.

clampValue [double](#)

The value to clamp.

waitTime [double](#)

The wait time after forcing.

pattern Pattern

The pattern to be executed during the test.

stops [int](#)

The number of stops executing a strobe on the instrument outside of the pattern - need to match with the pattern

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified **config**. For all pins specified in the **pinList** it disconnects any pin electronics, connects the dc path and turns on the gate.

The **Body** section applies a **forceVoltage** condition on all pins, sets the measurement mode, starts the pattern and performs a predefined number of current measurements synchronized by a Flag stop. Sets of measurements are made after Flag stop at the specified **waitTime**.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- any dc paths from pins in **pinList** are disconnected

Limitations

- support for non-uniform (mixed) instrument types in pinList not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(PinList pinList, double forceValue, double measureRange, double
clampValue, double waitTime, Pattern pattern,
                     int stops, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(pinList, nameof(pinList), out _pins);
        TheLib.Validate.Pattern(pattern, nameof(pattern), out _patternInfo);
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        TheLib.Validate.GreaterOrEqual(stops, 1, nameof(stops));
        _containsDigitalPins = _pins.ContainsFeature(InstrumentFeature.Digital);
        _patternInfo.SetFlags = (int)CpuFlag.A;
    }
}
```

```

if (ShouldRunPreBody) {
    TheLib.Setup.LevelsAndTiming.Apply(true);
    Services.Setup.Apply(setup);
    if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
    TheLib.Setup.Dc.Connect(_pins);
}

if (ShouldRunBody) {
    TheLib.Setup.Dc.SetForceAndMeter(_pins, TLibOutputMode.ForceVoltage, forceValue,
forceValue, clampValue, Measure.Current, measureRange);
    TheLib.Execute.Digital.StartPattern(_patternInfo);
    for (int i = 0; i < stops; i++) {
        TheLib.Execute.Digital.ContinueToConditionalStop(_patternInfo, () => {
            TheLib.Execute.Wait(waitTime);
            TheLib.Acquire.Dc.Strobe(_pins);
        });
    }
    TheLib.Execute.Digital.WaitPatternDone(_patternInfo);
    _pinsMeasuredSamples = TheLib.Acquire.Dc.ReadMeasuredSamples(_pins, stops);
}

if (ShouldRunPostBody) {
    TheLib.Setup.Dc.Disconnect(_pins);
    if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
    TheLib.Datalog.TestParametric(_pinsMeasuredSamples, forceValue, "V");
}
}

```

Class Static

Namespace: [Demo_CSRA.SupplyCurrent](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class Static : TestCodeBase
```

Overview

TestClass for all static supply current related TestMethods.

Platform Specifics

Uses per test-instance test class object persistence ([\[TestClass\(Creation.TestInstance\)\]](#) attribute).

Inheritance

[object](#) ← TestCodeBase ← Static

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
[TestCodeBase.TheHdw](#) , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
[TestCodeBase.ShouldRunPreBody](#) , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Baseline(PinList, double, double, double, double, string)

Executes a supply current test with the specified parameters.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Baseline(PinList pinList, double forceValue, double measureRange, double
clampValue, double waitTime, string setup = "")
```

Parameters

pinList PinList

List of pin or pin group names.

forceValue [double](#)

The value to force.

measureRange [double](#)

The range for measurement.

clampValue [double](#)

The value to clamp.

waitTime [double](#)

The wait time after forcing.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified **config**. For all pins specified in the **pinList** it disconnects any pin electronics, connects the dc path and turns on the gate.

The **Body** section applies a **forceVoltage** condition on all pins, sets the measurement mode and performs a current measurement on all pins in parallel after the specified **waitTime**.

The **PostBody** section restores the pin electronics connection for digital pins after gating off and disconnecting the dc path. Finally, a parametric datalog is logged.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- any dc paths from pins in `pinList` are disconnected

Limitations

- support for non-uniform (mixed) instrument types in `pinList` not yet available

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(PinList pinList, double forceValue, double measureRange, double
clampValue, double waitTime, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pins(pinList, nameof(pinList), out _pins);
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
        _containsDigitalPins = _pins.ContainsFeature(InstrumentFeature.Digital);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        if (_containsDigitalPins) TheLib.Setup.Digital.Disconnect(_pins);
        TheLib.Setup.Dc.Connect(_pins);
    }

    if (ShouldRunBody) {
        TheLib.Setup.Dc.SetForceAndMeter(_pins, TLibOutputMode.ForceVoltage, forceValue,
forceValue, clampValue, Measure.Current, measureRange);
        TheLib.Execute.Wait(waitTime);
        _meas = TheLib.Acquire.Dc.Measure(_pins);
    }

    if (ShouldRunPostBody) {
        TheLib.Setup.Dc.Disconnect(_pins);
        if (_containsDigitalPins) TheLib.Setup.Digital.Connect(_pins);
    }
}
```

```
TheLib.Datalog.TestParametric(_meas, forceValue, "V");  
}  
}
```

Namespace Demo_CSRA.Timing

Classes

[Frequency](#)

Class Frequency

Namespace: [Demo_CSRA.Timing](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class Frequency : TestCodeBase
```

Inheritance

[object](#) ← TestCodeBase ← Frequency

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
TestCodeBase.TheHdw , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
TestCodeBase.ShouldRunPreBody , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Baseline(Pattern, PinList, double, double, string, string, string)

Measures the frequency of the given digital pin(s) with the frequency counter while the pattern is executing.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Baseline(Pattern pattern, PinList pinList, double waitTime = 0, double
```

```
measureWindow = 0.01, string eventSource = "VOH", string eventSlope = "Positive", string
setup = "")
```

Parameters

pattern Pattern

The pattern to be executed during the test.

pinList PinList

Digital pin(s) to measure the frequency.

waitTime [double](#)

Optional. Time to wait between the pattern start and the start of the frequency measurement.

measureWindow [double](#)

Optional. Time to measure the frequency, longer measure times yield more accurate results.

eventSource [string](#)

Optional. The event source for the frequency counter (VOH, VOL or BOTH).

eventSlope [string](#)

Optional. The event slope for the frequency counter (Positive or Negative).

setup [string](#)

Optional. Setup to be applied before the pattern is run.

Details

Test Technique

- to be added

Implementation

The **validation** section creates the **pins** and **pattern** objects.

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified **setup**. For all pins specified in the **pinList** it connects the dc path and leaves the gate as it is.

The **Body** section configures the `measurePins` for a frequency read, starts the `pattern`, measures the frequency on the `measurePins` after the specified `wait` and then forces the `pattern` to halt. The pattern must be long enough for the specified `wait` plus the `measureWindow` and will be forced to halt immediately after the measurement is taken. This halt allows for the `pattern` to be implemented as an infinite loop or as a standard vector sequence, but does not allow for additional device configuration or processing after the frequency is measured.

The **PostBody** section datalogs the measured frequency.

Platform Specifics

Utilizes the frequency counter for the measurements. Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

- Only digital channels supported.

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void Baseline(Pattern pattern, PinList pinList, double waitTime = 0, double
measureWindow = 10 * ms, string eventSource = "VOH",
string eventSlope = "Positive", string setup = "") {

    if (TheExec.Flow.IsValidating) {
        TheLib.Validate.Pattern(pattern, nameof(pattern), out _pattern);
        TheLib.Validate.Pins(pinList, nameof(pinList), out _measurePins);
        TheLib.Validate.Enum(eventSource, nameof(eventSource), out _eventSource);
        TheLib.Validate.Enum(eventSlope, nameof(eventSlope), out _eventSlope);
        TheLib.Validate.InRange(measureWindow, 2.5 * ns, 10.7 * s, nameof(measureWindow));
        // 2.5ns and 10.7s are instrument limits on UP2200
        TheLib.Validate.InRange(waitTime, 0, 600, nameof(waitTime));
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
    }
}
```

```
if (ShouldRunBody) {
    TheLib.Setup.Digital.FrequencyCounter(_measurePins, measureWindow,
_eventSource, _eventSlope);
    TheLib.Execute.Digital.StartPattern(_pattern);
    TheLib.Execute.Wait(waitTime, true);
    _freqMeasure = TheLib.Acquire.Digital.MeasureFrequency(_measurePins);
    TheLib.Execute.Digital.ForcePatternHalt(_pattern);
}

if (ShouldRunPostBody) {
    TheLib.Datalog.TestParametric(_freqMeasure);
}
}
```

Namespace Demo_CSRA.Trim

Classes

[Digital](#)

Class Digital

Namespace: [Demo_CSRA.Trim](#)

Assembly: Demo_CSRA.dll

```
[TestClass(Creation.TestInstance)]
[Serializable]
public class Digital : TestCodeBase
```

Inheritance

[object](#) ← TestCodeBase ← Digital

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
TestCodeBase.TheHdw , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
TestCodeBase.ShouldRunPreBody , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

BinaryTarget(Pattern, PinList, int, int, double, bool, int, string)

Use binary search with target value to find which test step's output is closest to the target by capturing HRAM data.

```
[TestMethod]
[Steppable]
[CustomValidation]
```

```
public void BinaryTarget(Pattern pattern, PinList capPins, int from, int to, double minDelta, bool invertedOutput, int target, string setup = "")
```

Parameters

pattern Pattern

The pattern to run.

capPins PinList

The pins that are used to capture data through HRAM.

from [int](#)

The lower boundary of the search range.

to [int](#)

The upper boundary of the search range.

minDelta [double](#)

The minimum allowable difference between successive input values, used to determine when the search should stop.

invertedOutput [bool](#)

A flag indicating whether the output is inverted, affecting the search logic.

target [int](#)

The (numeric) target output value for which the corresponding input condition is searched.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`. Enable pattern threading for all pattern modules.

The **Body** section performs a binary search with target value on a hardcoded pattern (single pin) file with 255 modules. Each pattern module simulates the DUT response, and the search continues until it finds HRAM capture data that closest to the target.

The **PostBody** section restores the HRAM setup, and logs a parametric datalog.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

- none

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void BinaryTarget(Pattern pattern, PinList capPins, int from, int to, double
minDelta, bool invertedOutput, int target, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        if (!string.IsNullOrEmpty(pattern)) TheLib.Validate.Pattern(pattern,
nameof(pattern), out _pattern);
        TheLib.Validate.Pins(capPins, nameof(capPins), out _pins);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        for (int i = 0; i < _moduleCount; i++) {
            string name = _pattern.Name + $"":hram_mod{i}";
            PatternInfo patternTemp = new PatternInfo(name, true);
        }
    }

    if (ShouldRunBody) {
        var patSpec = new SiteVariant();
```

```

TheLib.Setup.Digital.ReadHram(8, CaptType.STV, TrigType.First, false, 0);
    _valuesBinary = TheLib.Acquire.Search.BinarySearch(from, to, minDelta,
invertedOutput, (modIndex) => {
    Site<int> result = new(-1);
    ForEachSite(site => {
        patSpec[site] = _pattern.Name + ":hram_mod{("int)modIndex[site]};";
    });
    TheHdw.PatternsPerSite(patSpec).Start();
    TheHdw.PatternsPerSite(patSpec).HaltWait();
    _readWords = TheLib.Acquire.Digital.ReadWords(_pins, 0, 8,
8, tlBitOrder.LsbFirst);
    ForEachSite(site => {
        result[site] = true ? (int)modIndex[site] : _readWords[0][site][0]; //
Simulate data process
    });
    return result;
},
target
);
}
}

if (ShouldRunPostBody) {
    TheLib.Setup.Digital.ReadHram(0, CaptType.None, TrigType.Never, false, 0);
    TheLib.Datalog.TestParametric(_valuesBinary);
}
}
}

```

BinaryTrip(Pattern, int, int, double, bool, string)

Use binary search with trip criteria to find where the test pattern result change from fail to pass.

```

[TestMethod]
[Steppable]
[CustomValidation]
public void BinaryTrip(Pattern pattern, int from, int to, double minDelta, bool
invertedOutput, string setup = "")

```

Parameters

pattern Pattern

The pattern to run.

`from int`

The lower boundary of the search range.

`to int`

The upper boundary of the search range.

`minDelta double`

The minimum allowable difference between successive input values, used to determine when the search should stop.

`invertedOutput bool`

A flag indicating whether the output is inverted, affecting the search logic.

`setup string`

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`.

The **Body** section performs a binary search on a hardcoded pattern (8 pins) file with 255 modules. Each pattern module simulates the DUT response, and the search continues until it finds input condition that make the test results to transition from fail to pass. If no such value is found, the function returns `-999` as a failure indicator.

The **PostBody** section logs a parametric datalog.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

- none

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void BinaryTrip(Pattern pattern, int from, int to, double minDelta, bool
invertedOutput, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        if (!string.IsNullOrEmpty(pattern)) TheLib.Validate.Pattern(pattern,
nameof(pattern), out _pattern);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
        for (int i = 0; i < _moduleCount; i++) {
            string name = _pattern.Name + $":mod{i}";
            PatternInfo patternTemp = new PatternInfo(name, true);
        }
    }

    if (ShouldRunBody) {
        var patSpec = new SiteVariant();
        _valuesBinary = TheLib.Acquire.Search.BinarySearch(from, to, minDelta,
invertedOutput, (modIndex) => {
            ForEachSite(site => {
                patSpec[site] = _pattern.Name + $":mod{(int)modIndex[site]}";
            });
            TheHdw.PatternsPerSite(patSpec).Start();
            TheHdw.PatternsPerSite(patSpec).HaltWait();
            return TheLib.Acquire.Digital.PatternResults();
        },
        patResult => patResult,
        _notFoundResult
    );
    }

    if (ShouldRunPostBody) {
        TheLib.Datalog.TestParametric(_valuesBinary);
    }
}
```

LinearFullTarget(Pattern, PinList, int, int, int, int, string)

Use linear full search with target value to find which test step's output is closest to the target by capturing HRAM data.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void LinearFullTarget(Pattern pattern, PinList capPins, int from, int to, int count,
int target, string setup = "")
```

Parameters

pattern Pattern

The pattern to run.

capPins PinList

The pins that are used to capture data through HRAM.

from [int](#)

The starting value of the linear input ramp.

to [int](#)

The stopping value of the linear input ramp.

count [int](#)

The total number of steps to execute.

target [int](#)

The (numeric) target output value to be searched.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`.

The **Body** section performs a full linear search with target value on a hardcoded pattern (single pin) file with 255 modules. Each pattern module simulates the DUT response, and once the search is completed, it finds HRAM capture data that closest to the target. If no such value is found, the function returns `-999` as a failure indicator.

The **PostBody** section restores the HRAM setup, and logs a parametric datalog.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

- none

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void LinearFullTarget(Pattern pattern, PinList capPins, int from, int to, int count,
int target, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        if (!string.IsNullOrEmpty(pattern)) TheLib.Validate.Pattern(pattern,
nameof(pattern), out _pattern);
        TheLib.Validate.Pins(capPins, nameof(capPins), out _pins);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
    }

    if (ShouldRunBody) {
        _measurementsTarget = [];
        TheLib.Setup.Digital.ReadHram(8, CaptType.STV, TrigType.First, false, 0);
    }
}
```

```

        double increment = TheLib.Acquire.Search.LinearFullFromToCount(from, to, count,
(modIndex) => {
    Site<int> result = new Site<int>(-1);
    TheHdw.Patterns(_pattern.Name + $"":hram_mod{modIndex}").Start();
    TheHdw.Digital.Patgen.HaltWait();
    _readWords = TheLib.Acquire.Digital.ReadWords(_pins, 0, 8,
8, tlBitOrder.LsbFirst);
    ForEachSite(site => {
        result[site] = true ? modIndex : _readWords[0][site][0]; // Simulate
data process
    });
    _measurementsTarget.Add(result);
});
_results = TheLib.Execute.Search.LinearFullProcess(_measurementsTarget, from,
increment, 0, target);
}

if (ShouldRunPostBody) {
    TheLib.Setup.Digital.ReadHram(0, CaptType.None, TrigType.Never, false, 0);
    TheLib.Datalog.TestParametric(_results);
}
}

```

LinearFullTrip(Pattern, int, int, int, string)

Use linear full search with trip criteria to find where the test pattern result change from fail to pass.

```

[TestMethod]
[Steppable]
[CustomValidation]
public void LinearFullTrip(Pattern pattern, int from, int to, int count, string setup = "")

```

Parameters

pattern Pattern

The pattern to run.

from [int](#)

The starting value of the linear input ramp.

to [int](#)

The stopping value of the linear input ramp.

count [int](#)

The total number of count to execute.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified **setup**.

The **Body** section performs a full linear search on a hardcoded pattern (8 pins) file with 255 modules. Each pattern module simulates the DUT response, and once the search is completed, it outputs the input conditions that cause the test results to transition from fail to pass. If no such value is found, the function returns **-999** as a failure indicator.

The **PostBody** section logs a parametric datalog.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

- none

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void LinearFullTrip(Pattern pattern, int from, int to, int count, string setup =
 "") {
```

```

if (TheExec.Flow.IsValidating) {
    if (!string.IsNullOrEmpty(pattern)) TheLib.Validate.Pattern(pattern,
nameof(pattern), out _pattern);
}

if (ShouldRunPreBody) {
    TheLib.Setup.LevelsAndTiming.Apply(true);
    Services.Setup.Apply(setup);
}

if (ShouldRunBody) {
    _measurementsTrip = [];
    double increment = TheLib.Acquire.Search.LinearFullFromToCount(from, to, count,
(modIndex) => {
    TheHdw.Patterns(_pattern.Name + $"":mod{modIndex}").Start();
    TheHdw.Digital.Patgen.HaltWait();
    _measurementsTrip.Add(TheLib.Acquire.Digital.PatternResults());
});
    _results = TheLib.Execute.Search.LinearFullProcess(_measurementsTrip, from,
increment, 0, _notFoundResult, condition => condition);
}

if (ShouldRunPostBody) {
    TheLib.Datalog.TestParametric(_results);
}
}

```

LinearStopTarget(Pattern, PinList, int, int, int, int, string)

Use linear stop search with target value to find which test step's output is closest to the target by capturing HRAM data.

```

[TestMethod]
[Steppable]
[CustomValidation]
public void LinearStopTarget(Pattern pattern, PinList capPins, int from, int to, int count,
int target, string setup = "")

```

Parameters

pattern Pattern

The pattern to run.

capPins PinList

The pins that are used to capture data through HRAM.

from [int](#)

The starting value of the linear input ramp.

to [int](#)

The stopping value of the linear input ramp.

count [int](#)

The total number of count to execute.

target [int](#)

The (numeric) target output value to be searched.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified **setup**.

The **Body** section performs a linear search with target value on a hardcoded pattern (single pin) file with 255 modules. Each pattern module simulates the DUT response, and the search continues until it finds HRAM capture data that closest to the target. If no such value is found, the function returns **-999** as a failure indicator.

The **PostBody** section restores the HRAM setup, and logs a parametric datalog.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

- none

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void LinearStopTarget(Pattern pattern, PinList capPins, int from, int to, int count,
int target, string setup = "") {

    if (TheExec.Flow.IsValidating) {
        if (!string.IsNullOrEmpty(pattern)) TheLib.Validate.Pattern(pattern,
nameof(pattern), out _pattern);
        TheLib.Validate.Pins(capPins, nameof(capPins), out _pins);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
    }

    if (ShouldRunBody) {
        TheLib.Setup.Digital.ReadHram(8, CaptType.STV, TrigType.First, false, 0);
        _values = TheLib.Acquire.Search.LinearStopFromToCount(from, to, count, 0,
_notFoundResult, (modIndex) => {
            Site<int> result = new Site<int>(-1);
            TheHdw.Patterns(_pattern.Name + $"":hram_{modIndex}").Start();
            TheHdw.Digital.Patgen.HaltWait();
            _readWords = TheLib.Acquire.Digital.ReadWords(_pins, 0, 8,
8, tlBitOrder.LsbFirst);
            ForEachSite(site => {
                result[site] = true ? modIndex : _readWords[0][site][0]; // Simulate
data process
            });
            return result;
        },
        target
    );
}
}
```

```
    if (ShouldRunPostBody) {
        TheLib.Setup.Digital.ReadHram(0, CaptType.None, TrigType.Never, false, 0);
        TheLib.Datalog.TestParametric(_values);
    }
}
```

LinearStopTrip(Pattern, int, int, int, string)

Use linear stop search with trip criteria to find where the test pattern result change from fail to pass.

```
[TestMethod]
[Steppable]
[CustomValidation]
public void LinearStopTrip(Pattern pattern, int from, int to, int count, string setup = "")
```

Parameters

pattern Pattern

The pattern to run.

from [int](#)

The starting value of the linear input ramp.

to [int](#)

The stopping value of the linear input ramp.

count [int](#)

The total number of count to execute.

setup [string](#)

Optional. The name of the setup set to be applied through the setup service.

Details

Test Technique

- to be added

Implementation

The **PreBody** section applies levels and timing from the test instance context. Optionally, applies the specified `setup`.

The **Body** section performs a linear search on a hardcoded pattern (8 pins) file with 255 modules. Each pattern module simulates the DUT response, and the search continues until it finds input condition that make the test pass. If no such value is found, the function returns `-999` as a failure indicator.

The **PostBody** section logs a parametric datalog.

Platform Specifics

Supports stepping capability for PreBody/Body/PostBody.

Pre Conditions

- none

Post Conditions

- none

Limitations

- none

Code Reference

```
[TestMethod, Steppable, CustomValidation]
public void LinearStopTrip(Pattern pattern, int from, int to, int count, string setup =
 "") {

    if (TheExec.Flow.IsValidating) {
        if (!string.IsNullOrEmpty(pattern)) TheLib.Validate.Pattern(pattern,
nameof(pattern), out _pattern);
    }

    if (ShouldRunPreBody) {
        TheLib.Setup.LevelsAndTiming.Apply(true);
        Services.Setup.Apply(setup);
    }

    if (ShouldRunBody) {
        _values = TheLib.Acquire.Search.LinearStopFromToCount(from, to, count, 0,
_notFoundResult, (modIndex) => {
            TheHdw.Patterns(_pattern.Name + $"":mod{modIndex}").Start();
            TheHdw.Digital.Patgen.HaltWait();
            return TheLib.Acquire.Digital.PatternResults();
        },
    }
}
```

```
    patResult => patResult
);
}

if (ShouldRunPostBody) {
    TheLib.Datalog.TestParametric(_values);
}
}
```

Class CustomerExtensions

Namespace: [Demo_CSRA](#)

Assembly: Demo_CSRA.dll

```
public static class CustomerExtensions
```

Inheritance

[object](#) ← CustomerExtensions

Inherited Members

[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

CustomerExtension(IDc, string, int)

```
public static void CustomerExtension(this ILib.ISetup.IDc dc, string argument1,  
int argument2)
```

Parameters

dc [ILib.ISetup.IDc](#)

argument1 [string](#)

argument2 [int](#)

Class ExecInterposeClass

Namespace: [Demo_CSRA](#)

Assembly: Demo_CSRA.dll

This class contains empty Exec Interpose functions.

```
[TestClass]
public class ExecInterposeClass : TestCodeBase
```

Inheritance

[object](#) ← TestCodeBase ← ExecInterposeClass

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
TestCodeBase.TheHdw , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
TestCodeBase.ShouldRunPreBody , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Remarks

They are here for convenience and are completely optional.

It is not necessary to delete them if they are not being used, nor is it necessary that they exist in the program.

Methods

OnAlarmOccurred(string)

Immediately after an alarm is detected, before it is reported.

```
[ExecInterpose_OnAlarmOccurred(255)]  
public static void OnAlarmOccurred(string alarmList)
```

Parameters

alarmList [string](#)

A tab-delimited string of alarm error messages.

OnFlowEnded()

Immediately after the flow has ended, before binning.

```
[ExecInterpose_OnFlowEnded(255)]  
public static void OnFlowEnded()
```

OnPostShutDownSite()

Immediately after a site is disconnected.

```
[ExecInterpose_OnPostShutDownSite(255)]  
public static void OnPostShutDownSite()
```

Remarks

Use [TheExec.Sites.SiteNumber](#) to determine which site is being disconnected.

OnPreShutDownSite()

Immediately after a site binning has been determined, before it is disconnected.

```
[ExecInterpose_OnPreShutDownSite(255)]  
public static void OnPreShutDownSite()
```

Remarks

Use `TheExec.Sites.SiteNumber` to determine which site is being disconnected.

OnProgramEnded()

Immediately after the test program has completed, before "post-job reset".

```
[ExecInterpose_OnProgramEnded(255)]  
public static void OnProgramEnded()
```

Remarks

Note that any actions taken here with respect to modification of binning will affect the binning sent to the Operator Interface, but will not affect the binning reported in Datalog.

OnProgramFailedValidation()

Immediately at the conclusion of the validation process. Called only if validation fails.

```
[ExecInterpose_OnProgramFailedValidation(255)]  
public static void OnProgramFailedValidation()
```

OnProgramLoaded()

Immediately after the test program has been loaded successfully.

```
[ExecInterpose_OnProgramLoaded(255)]  
public static void OnProgramLoaded()
```

OnProgramStarted()

Immediately after "pre-job reset" before each run of the test program starts.

```
[ExecInterpose_OnProgramStarted(255)]  
public static void OnProgramStarted()
```

Remarks

Note that "first run" actions can be enclosed in:

```
if (TheExec.ExecutionCount == 0) ...
```

OnProgramValidated()

Immediately at the conclusion of the validation process. Called only if validation succeeds.

```
[ExecInterpose_OnProgramValidated(255)]  
public static void OnProgramValidated()
```

OnValidationStart()

Immediately at the beginning of the validation process.

```
[ExecInterpose_OnValidationStart(255)]  
public static void OnValidationStart()
```

Class SetupLoadClass

Namespace: [Demo_CSRA](#)

Assembly: Demo_CSRA.dll

```
[TestClass]
public class SetupLoadClass : TestCodeBase
```

Inheritance

[object](#) ← TestCodeBase ← SetupLoadClass

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
TestCodeBase.TheHdw , [TestCodeBase.TheProgram](#) , [TestCodeBase.FlowDomains](#) ,
TestCodeBase.ShouldRunPreBody , [TestCodeBase.ShouldRunBody](#) , [TestCodeBase.ShouldRunPostBody](#) ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

MeasAction(PatternInfo, int)

```
public static List<PinSite<double>> MeasAction(PatternInfo patt, int stops)
```

Parameters

patt [PatternInfo](#)

stops [int](#)

Returns

[List](#) <PinSite<[double](#) >>

SetupLoad()

```
[TestMethod]  
public void SetupLoad()
```

Class Showcase

Namespace: [Demo_CSRA](#)

Assembly: Demo_CSRA.dll

```
[TestClass]
public class Showcase : TestCodeBase
```

Inheritance

[object](#) ← TestCodeBase ← Showcase

Inherited Members

[TestCodeBase.HandleUntrappedError\(Exception\)](#) , [TestCodeBase.AbortTest\(\)](#) ,
[TestCodeBase.ForEachSite\(Action<int>, tlSiteType, Func<Exception, bool>\)](#) ,
[TestCodeBase.CreateArray<T>\(int, string\)](#) , [TestCodeBase.CreateArray<T>\(int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(int, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, string\)](#) ,
[TestCodeBase.CreateArray<T>\(Func<T>, int, int, int, string\)](#) ,
[TestCodeBase.CreateLazyArray<T>\(params int\[\]\)](#) , [TestCodeBase.DebugBreak\(\)](#) , [TestCodeBase.TheExec](#) ,
TestCodeBase.TheHdw , TestCodeBase.TheProgram , TestCodeBase.FlowDomains ,
TestCodeBase.ShouldRunPreBody , TestCodeBase.ShouldRunBody , TestCodeBase.ShouldRunPostBody ,
[object.ToString\(\)](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#)

Methods

Baseline(Pattern, Pattern, string)

```
[TestMethod]
[Steppable]
[CustomValidation]
public void Baseline(Pattern patternA, Pattern patternB, string setup = "")
```

Parameters

patternA Pattern

patternB Pattern

setup string ↗

UseExtensionMethod()

```
[TestMethod]  
public void UseExtensionMethod()
```

Behavior Service - Features

This page lists known BehaviorService features for reference and to encourage consistent naming and usage patterns across projects. It is not the responsibility of the BehaviorService itself to define or track these — each client is free to define and manage its own set of features as needed.

Feature	Type	Description
Datalog.Parametric.OfflinePassResults	bool	use always-passing results for offline datalogs

TheLib

Acquire

- Dc
 - `public PinSite<double> Measure(Pins pins, Measure? meterMode = null)`
 - `public PinSite<double> Measure(Pins pins, int sampleSize, double? sampleRate = null, Measure? meterMode = null)`
 - `public PinSite<Samples<double>> MeasureSamples(Pins pins, int sampleSize, double? sampleRate = null, Measure? meterMode = null)`
 - `public PinSite<double> ReadCaptured(Pins pins, string signalName)`
 - `public PinSite<Samples<double>> ReadCapturedSamples(Pins pins, string signalName)`
 - `public PinSite<double> ReadMeasured(Pins pins, int sampleSize, double? sampleRate = null)`
 - `public PinSite<Samples<double>> ReadMeasuredSamples(Pins pins, int sampleSize, double? sampleRate = null)`
 - `public void Strobe(Pins pins)`
 - `public void StrobeSamples(Pins pins, int sampleSize, double? sampleRate = null)`
- Digital
 - `public Site<bool> PatternResults()`
 - `public PinSite<Samples<int>> Read(Pins pins, int startIndex = 0, int cycle = 0)`
 - `public PinSite<Samples<int>> ReadWords(Pins pins, int startIndex, int length, int wordSize, tlBitOrder bitOrder)`

Datalog

- `public void TestParametric(Site<int> result, double forceValue = 0, string forceUnit = "")`
- `public void TestParametric(Site<double> result, double forceValue = 0, string forceUnit = "")`
- `public void TestParametric(PinSite<int> result, double forceValue = 0, string forceUnit = "")`
- `public void TestParametric(PinSite<double> result, double forceValue = 0, string forceUnit = "")`
- `public void TestParametric(Site<Samples<int>> result, double forceValue = 0, string forceUnit = "", bool sameLimitForAllSamples = false)`
- `public void TestParametric(Site<Samples<double>> result, double forceValue = 0, string forceUnit = "", bool sameLimitForAllSamples = false)`
- `public void TestParametric(PinSite<Samples<int>> result, double forceValue = 0, string forceUnit = "", bool sameLimitForAllSamples = false)`
- `public void TestParametric(PinSite<Samples<double>> result, double forceValue = 0, string forceUnit = "", bool sameLimitForAllSamples = false)`

- `public void TestFunctional(Site<bool> result, string pattern = "")`

Execute

- Dc
 - `public PinSite<double> CalcResistance(PinSite<double> voltage, PinSite<double> current)`
 - `public PinSite<double> CalcResistance(PinSite<double> voltage, PinSite<double> current, PinSite<double> voltage2)`
 - `public PinSite<double> CalcResistance(PinSite<double> voltage1, PinSite<double> current1, PinSite<double> voltage2, PinSite<double> current2)`
- Digital
 - `public void StartPattern(PatternInfo patternInfo)`
 - `public void StartPattern(SiteVariant sitePatterns)`
 - `public void RunPattern(PatternInfo patternInfo)`
 - `public void RunPattern(SiteVariant sitePatterns)`
 - `public void WaitPatternDone(PatternInfo patternInfo)`
 - `public void ForcePatternHalt(PatternInfo patternInfo)`
 - `public void ForcePatternHalt()`
 - `public List<PinSite<double>> RunPatternConditionalStop(PatternInfo pattern, int numberofStops, Func<PatternInfo, int, List<PinSite<double>>> func)`
 - `public void ContinueToConditionalStop(PatternInfo pattern, Action action)`
- `public void Wait(double time, bool staticWait = false, double timeout = 100 * ms)`
- `public void CallByName(string name, string args)`

Setup

- Ac
 - `public void Connect(string pins)`
 - `public void Disconnect(string pins)`
- Dc
 - `public void ConnectAllPins()`
 - `public void Connect(Pins pins, bool? gateOn = null)`
 - `public void Disconnect(Pins pins, bool? gateOn = null)`
 - `public void Force(string pins, TLibOutputMode mode, double forceValue, double forceRange, double clampValue)`
 - `public void ForceI(Pins pins, double forceCurrent, double? clampVoltage = null)`
 - `public void ForceI(Pins pins, double forceCurrent, double clampVoltage, double currentRange, TLibOutputMode? outputMode = null, double? voltageRange = null)`
 - `public void ForceV(Pins pins, double forceVoltage)`
 - `public void ForceV(Pins pins, double forceVoltage, double voltageRange, double currentRange)`

- public void ForceV(Pins pins, double forceVoltage, double voltageRange, double currentRange, double? currentClamp = null, TLibOutputMode? outputMode = null)
- public void ForceHiZ(Pins pins, double? clampValue = null)
- public void SetMeter(Pins pins, Measure meterMode, double rangeValue, double? filterValue = null, double? hardwareAverage = null, double? outputRangeValue = null)
- Digital
 - public void Connect(Pins pins)
 - public void Disconnect(Pins pins)
 - public void ReadAll()
 - public void ReadFails()
 - public void ReadStoredVectors()
 - public void ReadHram(int captureLimit, CaptType captureType, TrigType triggerType, bool waitForEvent, int preTriggerCycleCount)
- Rf
 - public void Connect(string pins)
 - public void Disconnect(string pins)
- public void ApplyLevelsTiming()

Validation

- public Func<PatternInfo, int, List<PinSite<double>>> SetStopAction(string name)

Csra

Adc

- Ramp
 - Baseline
- Histogram
 - Baseline
- Dynamic
 - Baseline

Characterization

Continuity

- Parametric
 - Parallel(string pinList, double current, double clampVoltage, double voltageRange, double waitTime, string config = "")
 - Serial
- Functional
 - Baseline
- Supply
 - Baseline(string pinList, double forceVoltage, double currentRange, double waitTime, string config = "")
- Kelvin
 - Baseline

Dac

- Ramp
 - Baseline
- Histogram
 - Baseline
- Dynamic
 - Baseline

Functional

- Pattern
 - Baseline(string pattern, string config = "")
- Read
 - Baseline(string pattern, string readPins, int startIndex, int bitLength, int wordLength, bool msbFirst, bool testFunctional, bool testValues, string config = "")

- Scan

Leakage

- Parallel
 - Baseline(string pinList, double voltage, double currentRange, double waitTime, string config = "")
 - Preconditioning(string pattern, string measurePins, double voltage, double currentRange, double waitTime, string configAction = "")

Memory

- Mbist
 - Baseline
 - Stress
- Repair
 - Baseline
- Retention
 - Baseline

OneTimeProgramming

Parametric

- SingleCondition
 - Baseline
 - PreconditionPattern
 - PatternHandshake
- MultiCondition
 - Baseline
 - PreconditionPattern
 - PatternHandshake

Resistance

- Contact
 - OnePinDeltaForceDeltaMeasure(string forcePin, string forceMode, double forceFirstValue, double forceSecondValue, double clampValueOfForcePin, double measureFirstRange, double measureSecondRange, double waitTime = 0.0, string config = "")
- RdsOn
 - OnePinOneForceMeasure(string forcePin, string forceMode, double forceValue, double measureRange, double waitTime = 0, string labelOfStoredVoltage = "", string config = "")

- """)
- TwoPinsOneForceOneMeasure(string forcePin, string forceMode, double forceValue, double clampValueOfForcePin, string measurePin, double measureRange, double waitTime = 0, string labelOfStoredVoltage = "", string config = "")
- TwoPinsDeltaForceDeltaMeasure(string forcePin, string forceMode, double forceFirstValue, double forceSecondValue, double clampValueOfForcePin, string measurePin, double measureFirstRange, double measureSecondRange, double waitTime = 0, string config = "")
- ThreePinsOneForceTwoMeasure(string forcePin, double forceCurrentPin, double clampValueOfForcePin, string measureFirstPin, double measureRangeFirstPin, string measureSecondPin, double measureRangeSecondPin, double waitTime = 0, string config = "")
- FourPinsTwoForceTwoMeasure(string forceFirstPin, double forceValueFirstPin, double clampValueOfForceFirstPin, string forceSecondPin, double forceValueSecondPin, double clampValueOfForceSecondPin, string measureFirstPin, string measureSecondPin, double measureRangeFirstPin, double measureRangeSecondPin, double waitTime = 0, string config = "")

Rf

- Power
- Noise
- Imd
- Modulation

Search

SupplyCurrent

- Dynamic
 - Baseline(string pinList, double forceValue, double measureRange, double clampValue, double waitTime, string pattern, int stops, string config = "")
- Static
 - Baseline(string pinList, double forceValue, double measureRange, double clampValue, double waitTime, string config = "")

Timing

- Jitter
- PropagationDelay
- Frequency
 - Baseline

- EdgeSearch
- RiseFallTime
- EdgeCount

Trim

SetupService: DCVI Features

The DCVI is a relatively complex instrument, with language nested in multiple levels.

Level 1 Nodes

TheHdw.Dcvi Features

Node	Type	Access	Implementation	MS status
EnableLevelSequence	bool	{ get; set; }	boolean flag	?
HighVoltageComplianceRangeEnabled	bool	{ get; set; }	boolean flag	?
PatternRestartOptimizationEnabled	bool	{ get; set; }	boolean flag	?
Pins(string)	DriverDCVIPins	{ get; }	sub-node	:mechanic:

Level 2 Nodes

TheHdw.Dcvi.Pins() Features

Node	Type	Access	Implementation	MS status
Alarm[tDCVIAlarm]	IDCVIAlarmIndexer	{ get; set; }	enum	?
BleederResistor	DriverDCVIBleederResistor	{ get; }	sub-node	✓
ComplianceRange_Positive	IDiscreteValue	{ get; }	:double:	✓
ComplianceRange_Negative	IDiscreteValue	{ get; }	:double:	✓
Calibration	DriverDCVICalibration	{ get; }	?	?

Node	Type	Access	Implementation	MS status
Capture	DriverDCVICapture	{ get; }	sub-node	?
Connect(tlDCVIConnectWhat)	void	Method	enum	✓
Connected	tlDCVIConnectWhat	{ get; }	enum	✓
Current	IDoublePerSite	{ get; }	IDoublePerSite	✓
CurrentRange	IDiscreteAutoValue	{ get; }	IDiscreteAutoValue	✓
ExternalModulationInput	bool	{ get; set; }	bool	?
FoldCurrentLimit	DriverDCVIFoldCurrentLmt	{ get; }	sub-node	✓
Gate	tlDCVIGate	{ get; set; }	enum	✓
KelvinAlarm(tlDCVIKelvinAlarm)	DriverDCVIKelvinAlarm	Method	?	?
Meter	DriverDCVIMeter	{ get; }	sub-node	?
Mode	tlDCVIMode	{ get; set; }	enum	✓
NominalBandwidth	DriverDCVINominalBandWidth	{ get; }	quantified float	✓
PSets	DriverDCVIPSetCollection	{ get; }	?	?
PulsedPower	DriverDCVIPulsedPower	{ get; }	sub-node	🔒
Reset	void	Method	?	?

Node	Type	Access	Implementation	MS status
Snubber	DriverDCVISnubber	{ get; }	?	?
SoftKelvin	DriverDCVISoftKelvin	{ get; }	?	?
Source	DriverDCVISource	{ get; }	sub-node	?
SpikeCheck	DriverDCVISPikeCheck	{ get; }	?	?
Voltage	IDoublePerSite	{ get; }	IDoublePerSite	✓
VoltageRange	IDiscreteAutoValue	{ get; }	IDiscreteAutoValue	✓

Level 3 Nodes

TheHdw.DcvI.Pins().BleederResistor Features

Node	Type	Access	Implementation	MS status
Mode	t1DCVIBleederResistor	{ get; set; }	enum	✓
CurrentLoad	double	{ get; set; }	double	✓

TheHdw.DcvI.Pins().FoldCurrentLimit Features

Node	Type	Access	Implementation	MS status
Behavior	t1DCVIFolderCurrentLimitBehavior	{ get; set; }	enum	✓
Timeout	IContinuousValue	{ get; }	IContinuousValue	✓

TheHdw.DcvI.Pins().PulsedPower Features

Node	Type	Access	Implementation	MS status
PulseVoltage	IContinuousValue	{ get; set; }	double	
PulseCurrent	IContinuousValue	{ get; set; }	double	
PulseDelay	IContinuousValue	{ get; set; }	double	
PulseDuration	IContinuousValue	{ get; set; }	double	

TheHdw.Dcvi.Pins().Capture Features

TheHdw.Dcvi.Pins().Meter Features

TheHdw.Dcvi.Pins().Source Features

SetupService: DCVS Features

The DCVS is a relatively complex instrument, with language nested in multiple levels.

Level 1 Nodes

TheHdw.Dcvs Features

Node	Type	Access	Implementation	MS status
EnableUndefinedPSetsCheckInPattern	bool	{ get; set; }	boolean flag	?
Pins	DriverDCVSPins	{ get; }	sub-node	:mechanic:

Level 2 Nodes

TheHdw.Dcvs.Pins() Features

Node	Type	Access	Implementation	MS status
BandwidthSetting	IDiscreteValue	{ get; }	IDiscreteValue (quantified)	?
BleederResistor	t1DCVSOnOffAuto	{ get; set; }	enum	✓
Capture	DriverDCVSCapture	{ get; }	sub-node	?
Connect	void	Method	enum	✓
Connected	t1DCVSConnectWhat	{ get; }	enum	✓
CurrentLimit	DriverDCVSCurrentLimit	{ get; }	sub-node	:mechanic:
CurrentRange	IDiscreteValue	{ get; }	IDiscreteValue (quantified)	✓
Disconnect	void	Method	enum	✓

Node	Type	Access	Implementation	MS status
Gate	bool	{ get; set; }	bool	✓
get_KelvinAlarm(tlDCVSHat)	DriverDCVSKelvinAlarm	Method	?	?
Meter	DriverDCVSMeter	{ get; }	sub-node	?
Mode	tlDCVSMode	{ get; set; }	enum	✓
PSets	DriverDCVSPSetCollection	{ get; }	?	?
Reset	void	Method	enum	?
Source	DriverDCVSSource	{ get; }	sub-node	?
SpikeCheck	DriverDCVSSpikeCheck	{ get; }	?	?
Voltage	DriverDCVSVoltage	{ get; }	?	✓
VoltageRange	IDiscreteAutoValue	{ get; }	IDiscreteAutoValue (quantified)	✓

Level 3 Nodes

TheHdw.Dcvs.Pins().Capture Features

Node	Type	Access	Implementation	MS status
SampleRate	IContinuousValue	{ get; }	IContinuousValue	?
SampleSize	IContinuousValue	{ get; }	IContinuousValue	?

TheHdw.Dcvs.Pins().CurrentLimit Features

Node	Type	Access	Implementation	MS status
Sink	DriverDCVSCurrentLimitSink	{ get; }	sub-node	?
Source	DriverDCVSCurrentLimitSource	{ get; }	sub-node	:mechanic:

TheHdw.Dcvs.Pins().Meter Features

Node	Type	Access	Implementation	MS status
CurrentRange	IDiscreteValue	{ get; }	IDiscreteValue	?
Filter	IDiscreteFilterValue	{ get; }	IDiscreteFilterValue	?
Mode	t1DCVSMeterMode	{ get; set; }	enum	?

TheHdw.Dcvs.Pins().PSets Features

Node	Type	Access	Implementation	MS status
xxx	xxx	xxx	?	?

TheHdw.Dcvs.Pins().Source Features

Node	Type	Access	Implementation	MS status
SampleRate	IContinousValue	{ get; }	IContinousValue	?

Level 4 Nodes

TheHdw.Dcvs.Pins().CurrentLimit.Sink Features

Node	Type	Access	Implementation	MS status
FoldLimit	DriverDCVSCurrentLimitSinkFold	{ get; }	sub-node	?
OverloadLimit	DriverDCVSCurrentLimitSinkOverload	{ get; }	sub-node	?
SetLimitLevels	void	Method	double	?
SetLimitTimeouts	void	Method	double	?

TheHdw.Dcvs.Pins().CurrentLimit.Source Features

Node	Type	Access	Implementation	MS status
FoldLimit	DriverDCVSCurrentLimitSourceFold	{ get; }	sub-node	:mechanic:
OverloadLimit	DriverDCVSCurrentLimitSourceOverload	{ get; }	sub-node	:mechanic:
SetLimitLevels	void	Method	double	?
SetLimitTimeouts	void	Method	double	?

Level 5 Nodes

TheHdw.Dcv.Pins().CurrentLimit.Source.FoldLimit Features

Node	Type	Access	Implementation	MS status
Behavior	t1DCVSCurrentLimitBehavior	{ get; set; }	enum	?
Level	IContinuousValuePerSite	{ get; }	IContinuousValuePerSite	✓
Timeout	IContinuousValue	{ get; }	IContinuousValue	?

TheHdw.Dcv.Pins().CurrentLimit.Source.OverloadLimit Features

Node	Type	Access	Implementation	MS status
Behavior	t1DCVSCurrentLimitBehavior	{ get; set; }	enum	?
Level	IContinuousValuePerSite	{ get; }	IContinuousValuePerSite	✓
Timeout	IContinuousValue	{ get; }	IContinuousValue	?

TheHdw.Dcv.Pins().CurrentLimit.Sink.FoldLimit Features

Node	Type	Access	Implementation	MS status
Behavior	t1DCVSCurrentLimitBehavior	{ get; set; }	enum	?
Level	IContinuousValuePerSite	{ get; }	IContinuousValuePerSite	✓
Timeout	IContinuousValue	{ get; }	IContinuousValue	?

TheHdw.Dcv.Pins().CurrentLimit.Sink.OverloadLimit Features

Node	Type	Access	Implementation	MS status
Behavior	tlDCVSCurrentLimitBehavior	{ get; set; }	enum	?
Level	IContinousValuePerSite	{ get; }	IContinousValuePerSite	✓
Timeout	IContinousValue	{ get; }	IContinousValue	?

SetupService: Digital (PinElectronics) Features

The PE is a relatively complex instrument, with language nested in multiple levels.

Level 1 Nodes

TheHdw.Digital Features

Node	Type	Access	Implementation	MS status
Pins	DriverDigitalPins	Method	sub-node	✓

Level 2 Nodes

TheHdw.Digital.Pins() Features

Node	Type	Access	Implementation	MS status
Connect	void	Method	()	✓
Connected	bool	{ get; }	boolean flag	✓
Disconnect	void	Method	()	✓
InitState	ChInitState	{ get; set; }	enum	✓
Levels	DriverDigPinsLevels	{ get; }	sub-node	✓
StartState	ChStartState	{ get; set; }	enum	✓

Level 3 Nodes

TheHdw.Digital.Pins().Levels Features

Node	Type	Access	Implementation	MS status
DriverMode	t1DriverMode	{ get; set; }	enum	✓
Value	IDigitalPinsValueIndexer	{ get; set; }	sub-node	✓

Level 4 Nodes

TheHdw.Digital.Pins().Levels.Value Features

Node	Type	Access	Implementation	MS status
Value_Vih	double	{ get; set; }	double	✓
Value_Vil	double	{ get; set; }	double	✓
Value_Vt	double	{ get; set; }	double	✓
Value_Vcl	double	{ get; set; }	double	✓
Value_Vch	double	{ get; set; }	double	✓
Value_Ioh	double	{ get; set; }	double	✓
Value_Iol	double	{ get; set; }	double	✓
Value_Voh	double	{ get; set; }	double	✓
Value_Vol	double	{ get; set; }	double	✓

SetupService: PPMU Features

The PPMU is a moderately complex instrument, with language nested in multiple levels.

Level 1 Nodes

TheHdw.PPMU Features

Node	Type	Access	Implementation
AllowPPMUFuncRelayConnection	void	Method	bool
HighAccuracyMeasureVoltage	DriverPPMUHighAccuracyMeasureVoltage	{ get; }	sub-node
MinimizeTransitionEnergy	bool	{ get; set; }	boolean flag
Pins	tlDriverPPMUPins	Method	sub-node
SetClampsVHi	void	Method	double
SetClampsVLo	void	Method	double
UseFlowLimits	bool	{ get; set; }	boolean flag

Level 2 Nodes

TheHdw.PPMU.HighAccuracyMeasureVoltage Features

Node	Type	Access	Implementation	MS status
Enabled	bool	{ get; set; }	boolean flag	?
SettlingTime	double	{ get; set; }	double	?

TheHdw.PPMU.Pins() Features

Node	Type	Access	Implementation	MS stat
ClampVHi	IContinuousValue	{ get; }	IContinuousValue	?

Node	Type	Access	Implementation	MS stat
ClampVLo	IContinuousValue	{ get; }	IContinuousValue	?
Connect	void	Method	boolean flag	✓
Current	IPPMUCurrentContinuousValue	{ get; }	IContinuousValue	?
Disconnect	void	Method	()	✓
ForceCurrentRange	IPPMUForceCurrentRangeDiscreteAutoValue	{ get; }	IDiscreteAutoValue	?
ForceI	void	Method	(params)	?
ForceV	void	Method	(params)	?
ForceVMeasureV	void	Method	(params)	?
Gate	t1OnOff	{ get; set; }	enum	✓
IsConnected	bool	{ get; }	boolean flag	✓
MeasureCurrentRange	PPMUMeasureCurrentRangeDiscreteAutoValue	{ get; }	IDiscreteAutoValue	?
Mode	t1PPMUMode	{ get; }	enum	?
Voltage	IPPMUVoltageContinuousValue	{ get; }	IContinuousValue	?

SetupService: Utility Features

Level 1 Nodes

TheHdw.Utility Features

Node	Type	Access	Implementation	MS status
EnableUtilityBitsReset	bool	{ get; set; }	boolean flag	?
Reset	void	Method	()	?
Threshold	double	{ get; set; }	double	?
Pins	tlDriverUtilityPins	Method	sub-node	:mechanic:

Level 2 Nodes

TheHdw.Utility.Pins Features

Node	Type	Access	Implementation	MS status
State	tlUtilBitState	{ set; }	enum	✓
States	IPinListData	Method	enum	?

SetupService: Setting Types

To fulfill the requirements for the **SetupService**, each data type is implemented as a specifically typed **Setting<T>**.

The table below lists all types needed for the implementation of DCVI, DCVS, PPMU, Digital and Utility as currently defined (commit #5164760dff54efec76ef2fc2823a08b9177ea04).

Base Types

Type	ImplementationName	MS status
double	SettingDouble	✓
int	SettingInt	✓
bool	SettingBool	✓
enum	Table Enums	✓

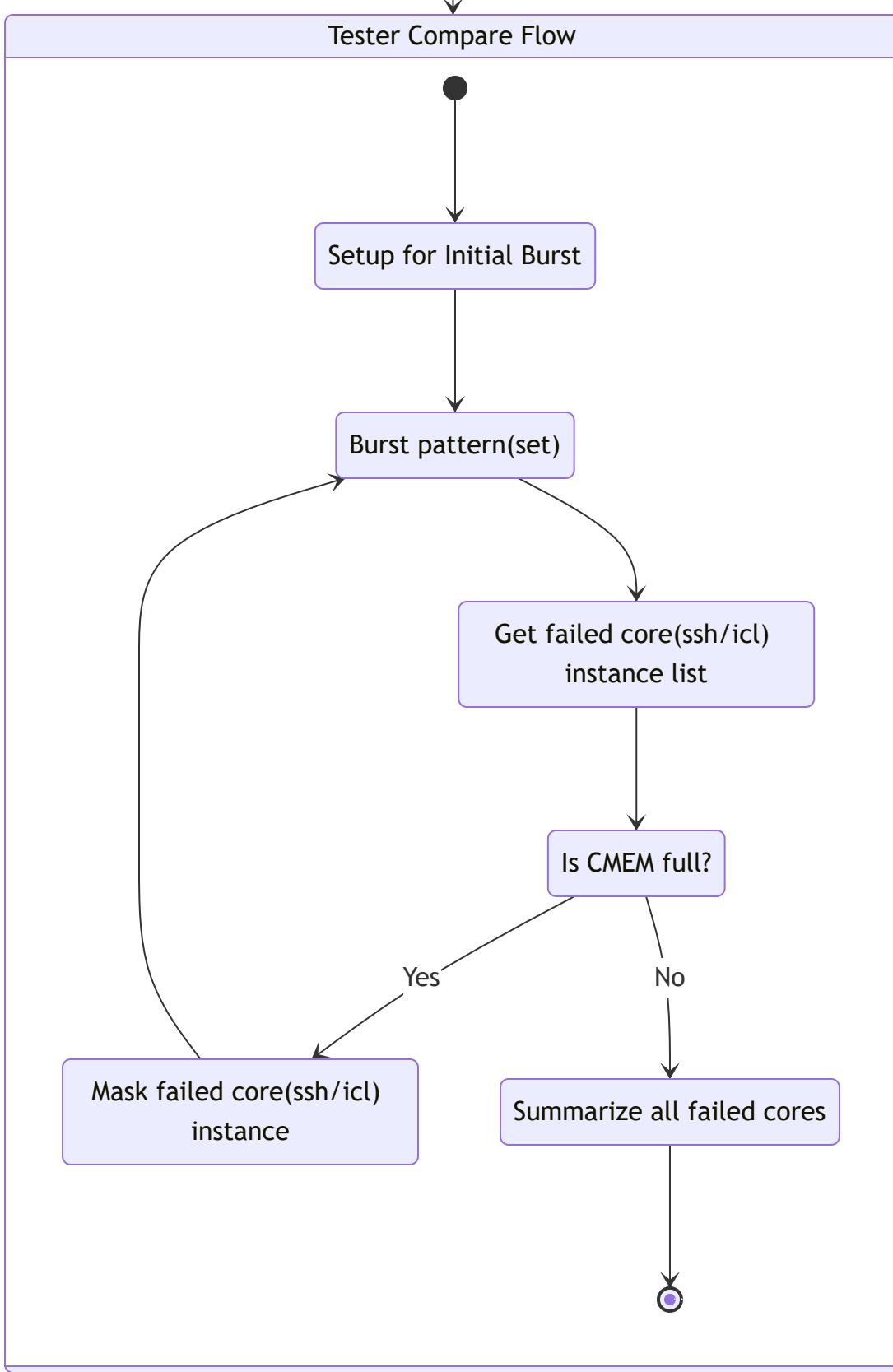
Enums

EnumType	ImplementationName	MS status
tlUtilityBitState	SettingUtilityBitState	✓
tlOnOff	SettingOnOff	✓
ChInitState	SettingChInitState	✓
ChStartState	SettingChStartState	✓
tlDriverMode	SettingDriverMode	✓
tlPPMUMode	SettingPPMUMode	✓
tlDCVIconnectWhat	SettingDCVIconnectWhat	✓
tlDCVGate	SettingDCVGate	✓
tlDCVIMode	SettingDCVIMode	✓
tlDCVIBleederResistor	SettingDCVIBleederResistor	✓
tlDCVIFoldCurrentLimitBehavior	SettingDCVIFoldCurrentLimitBehavior	✓

EnumType	ImplementationName	MS status
t1DCVSOnOffAuto	SettingDCVSOnOffAuto	✓
t1DCVSConnectWhat	SettingDCVSConnectWhat	✓
t1DCVSMode	SettingDCVSMode	✓
t1DCVSMeterMode	SettingDCVSMeterMode	✓
t1DCVSCurrentLimitBehavior	SettingDCVSCurrentLimitBehavior	✓

SSN Pattern's Special TestFlow

1. Non-Diagnosis TestFlow if all ssh are OCComp = off
(Tester Compare Flow)



1.1 Setup Initial Burst

```
// shall be encapsulated in C#RA, something like:  
// TheLib.Setup.ssn.SetupCmemForInitialBurst();  
TheHdw.Digital.CMEM.CentralFields = tlCMEMCaptureFields.AbsoluteCycle |  
tlCMEMCaptureFields.PatternName;  
TheHdw.Digital.CMEM.SetCaptureConfig(-1, CmemCaptType.Fail,  
tlCMEMCaptureSource.PassFailData, true, false);  
TheHdw.Digital.CMEM.CaptureLimitMode = tlDigitalCMEMCaptureLimitMode.Enable;  
TheHdw.Digital.CMEM.CaptureLimit = maxFailsPerPin;
```

1.2 Burst pattern(set)

```
TheHdw.Patterns(ssnPatternSet).ExecuteSet(tlPatternSetResultType.Functional);
```

1.3 Get failed core(ssh/icl) instance list

```
var ssnTcResults = TheHdw.Digital.Patgen.ReadScanNetworkResults();  
var failedCoreList = ssnTcResults.FailedCores;
```

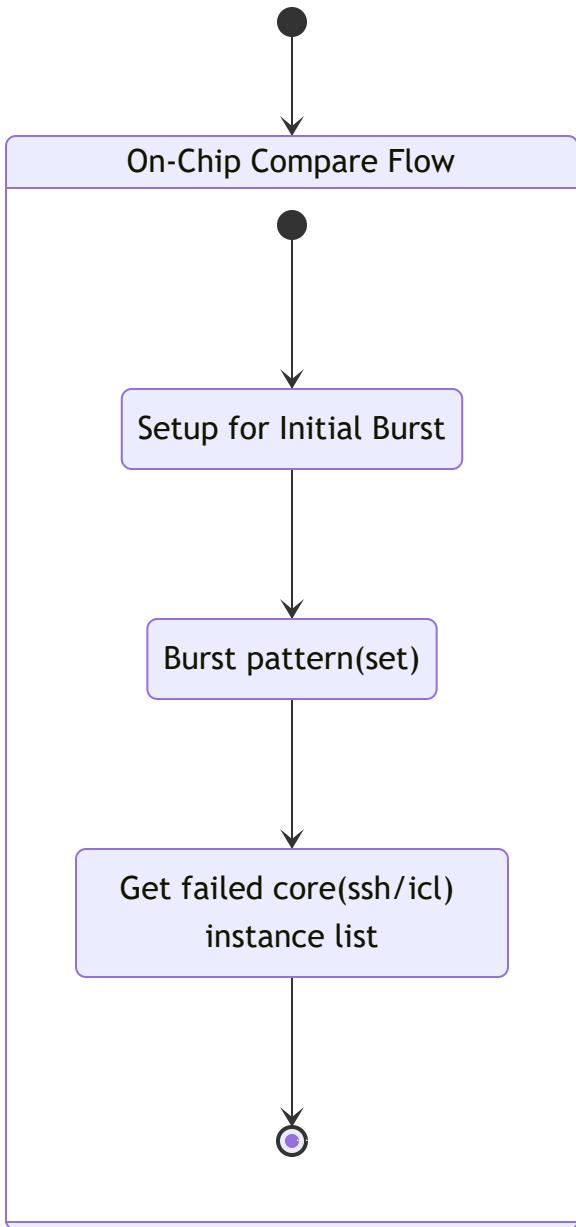
1.4 Is CMEM full? [1.5 Mask-n-Reburst](#), [1.6 Conclude](#)

1.5 Mask failed core(ssh/icl) instance and goto [1.2 burst pattern\(set\)](#)

```
var ssnPattern = TheHdw.Digital.ScanNetworks[ssnMapfileName];  
ssnPattern.CoreMasks.AddPerSite(failedCoreList);  
ssnPattern.CoreMasks.Apply();
```

1.6 TC Flow Complete

2. Non-Diagnosis TestFlow if all ssh are OCCOMP = on (On-Chip Compare Flow)



2.1 Setup Initial Burst

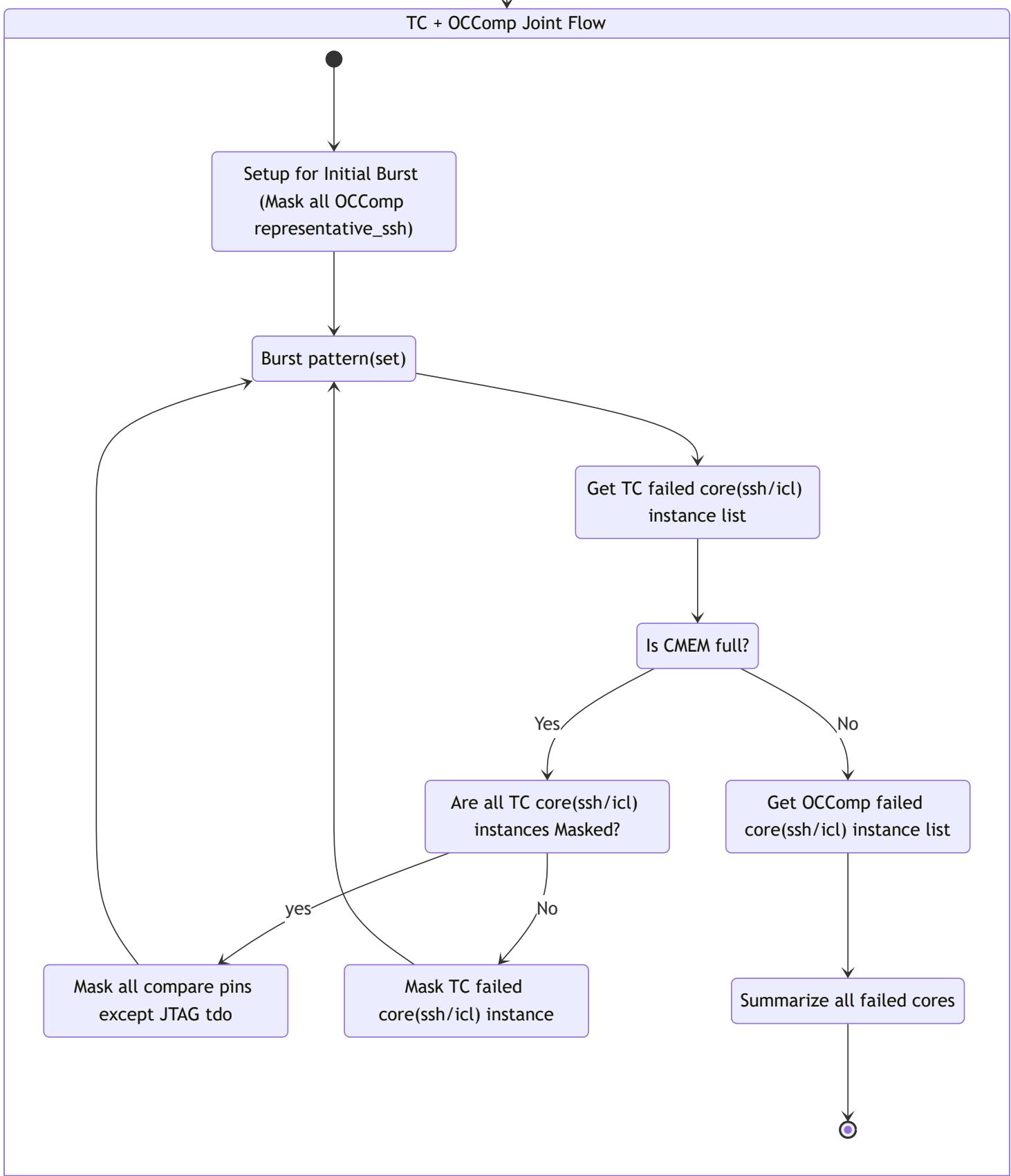
The difference between TC and OCCOMP is that in OCCOMP user only cares about sticky_bit status. (all the sticky_bits) So before the Initial Burst of the pattern, it is better to **Mask** all output pins except the JTAG tdo pin.

2.2 Burst Pattern(set)

2.3 Get failed core(ssh/icl) instance list (by sticky_bit on tdo pin)

2.4 OCCOMP Flow Complete

3. TC + OCCOMP Joint Flow (None-Diagnosis)



3.1 Setup Initial Burst

In this case, all representative_ssh instances shall be TC masked before Initial Burst since we don't want them to consume CMEM, and we don't need them to determine any OCCComp ssh-icl-instance's pass fail status.

3.2 Burst Pattern(set)

3.3 Get TC failed core(ssh/icl) instance list

3.4 Is CMEM full? [3.5 Mask-TC-n-Reburst](#), [3.6 Get OCCComp failed cores](#)

If CMEM is NOT full, it means that all sticky_bit fails on JTAG tdo pin are captured completely.

3.5 Mask failed core(ssh/icl) instance and goto [3.2 Burst pattern\(set\)](#)

```
var ssnPattern = TheHdw.Digital.ScanNetworks[ssnMapfileName];
ssnPattern.CoreMasks.AddPerSite(failedCoreList);
ssnPattern.CoreMasks.Apply();
```

in case that all TC core instances are masked and yet still the CMEM is full, it is possible that none-scan compares are eating up the CMEM. in which case we need to mask all output pins except the tdo pins just like what we do in [2.1 Setup for OCCComp Initial Burst](#)

3.6 Get OCCComp failed core(ssh/icl) instance list (by sticky_bit on tdo pin)

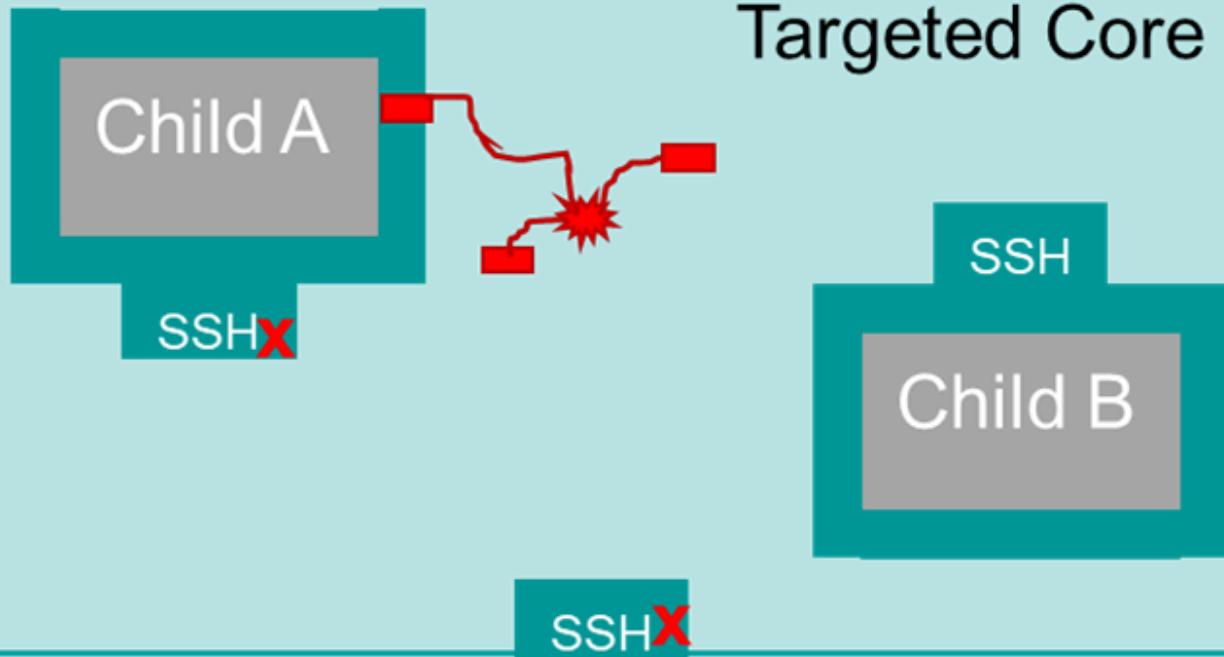
3.7 TC+OCCComp Joint Flow complete

4. Diagnosis Flow

Failures may be captured in:

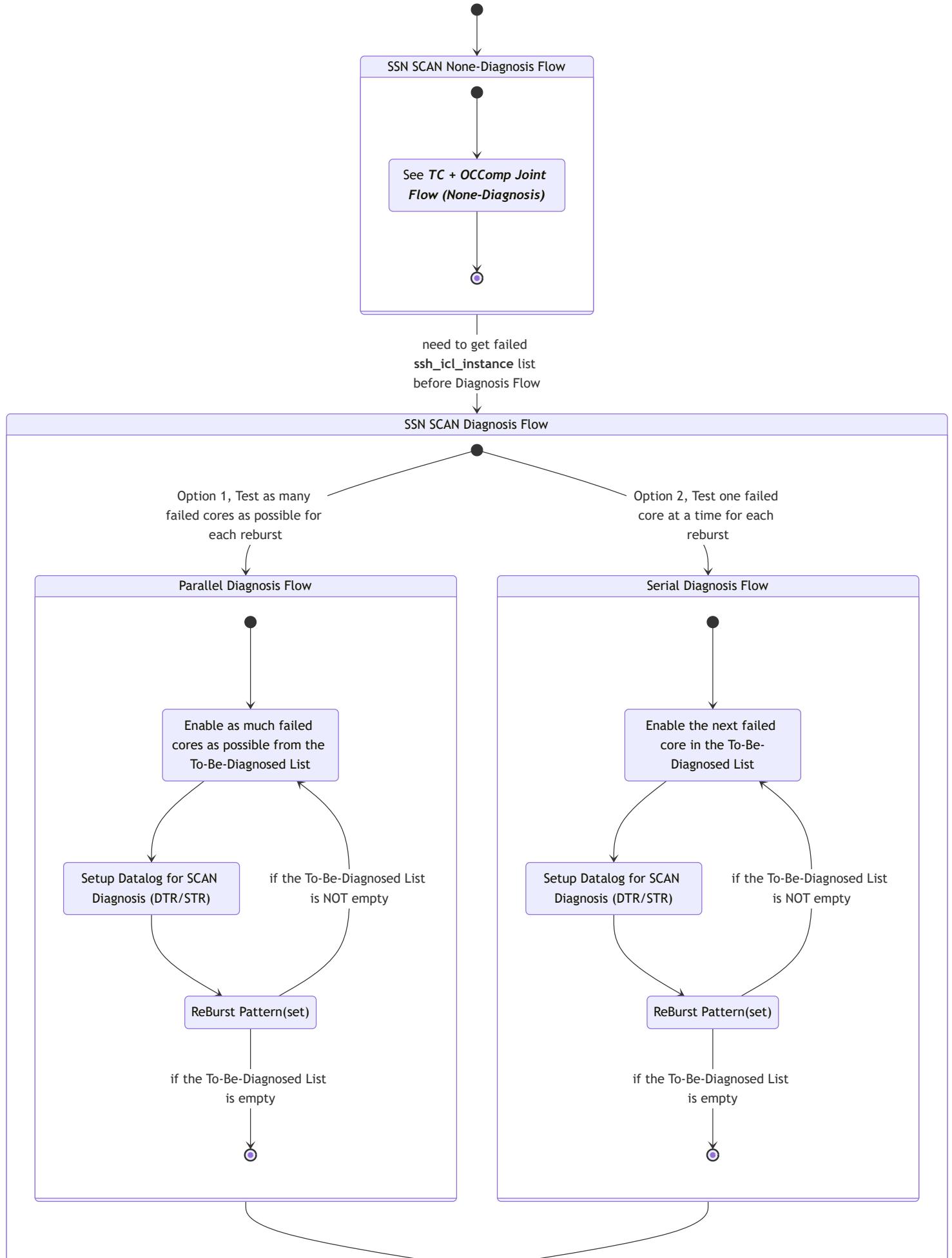
- Internal scan chains if target core
- Wrapper chain of child core

Targeted Core



Failures from targeted core and child cores must be diagnosed together

And since TC + OCComp Joint Flow can cover the first two cases, we will only implement the TC + OCComp Joint Flow. So, the Diagnosis Flow shall be:





4.1 Setup Device for Diagnosing Failed Core(s)

- Serial Reburst: one failed core at a time
- Parallel Reburst: as many failed cores as possible per burst

4.2 Setup Datalog for SCAN Diagnosis

DTR and CONDITION_LIST of STR need to be configured before bursting the pattern

4.3 Burst Pattern(set)

(nothing special)

SSN Slang Glossary

To address the traditional SCAN technology's sustainability pain points, SSN has created a lot of new concepts that does not exist before and a lot of acronyms along with them. The purpose of this cheat sheet is supposed to help users who are new to SSN to understand the slang that we speak.

Platform agnostic terms

Accumulated Status

- refer to [On-Chip Compare](#)
- Per-shift pass/fail status bits accumulated across all `ssh_icl_instances` of the same `capture_global_group`

`capture_global_group`

- refer to [representative ssh](#)
- `ssh_icl_instances` that share the same `capture_global_group` ID appear as only one `ssh_icl_instance` on the ssn bus.

Contribution Bits (`disable_contribution_bit`)

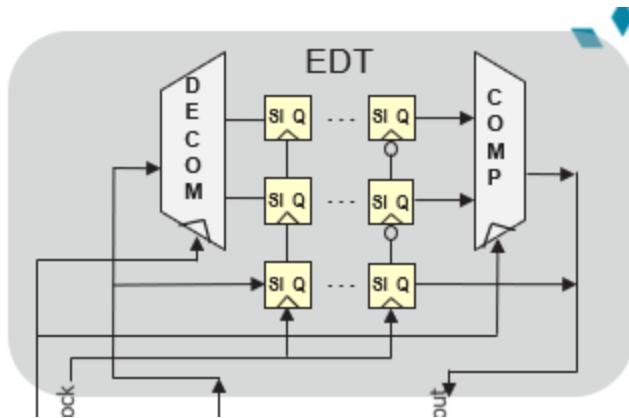
- A RW bit, if asserted (set to 1), that can prevent a `ssh_icl_instance`'s On-Chip compare logic to modify(contribute) the `Accumulated Status` slots.
- refer to [On-Chip Compare](#).

`core_instance`

- A core is often a function IP with established `EDT` networks.
- It can be assigned with one ssh instance or multiple if the core is too complex.
- you can consider a `core instance` as a group of `ssh_icl_instances`, note that the group may only contains one `ssh_icl_instance`.
- when testing, core instance need to be tested as a minimum unit. which means all `ssh_icl_instances` of a given `core instance` need to be enabled.

`EDT`

- Embedded Deterministic Test



ICL (icl)

- Instrument Connectivity Language

iJTAG

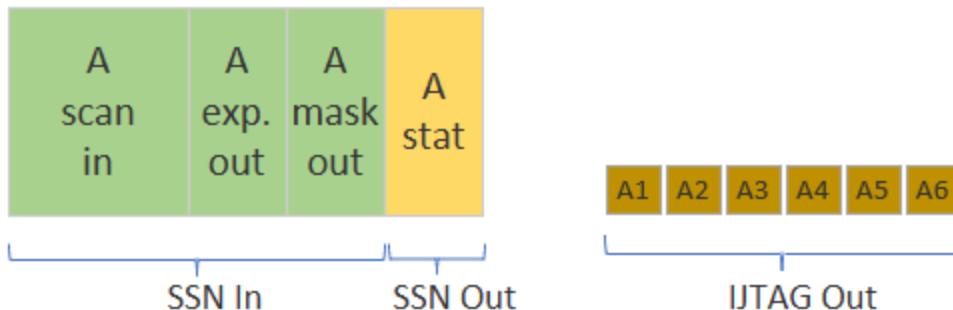
- IEEE 1687

On-Chip Clock (OCC)

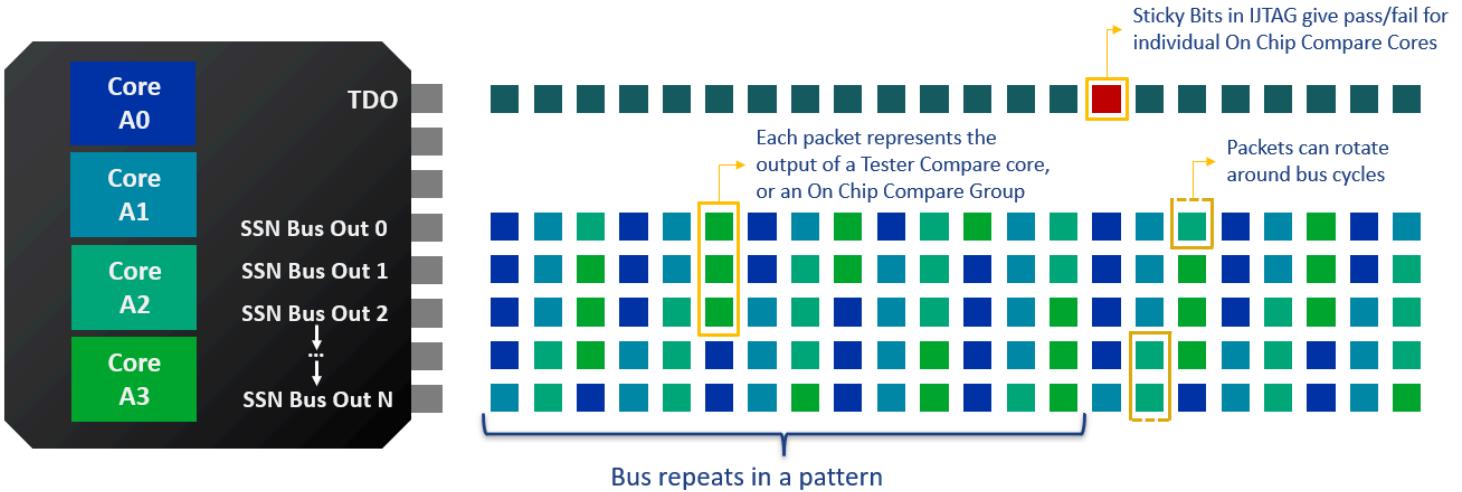
- this is to highlight that **occ** is typically referring to **On Chip Clock**, while **On-Chip Compare** uses **OCComp**

On-Chip Compare (OCComp)

- A ssh takes its scan in data as well as scan out data (expected and masked) from predictable preallocated **time slot** on the ssn bus and offload the mis-compare on a dedicated region on the preallocated **time slot** called **accumulated status** zone.



- Per-shift pass/fail status bits accumulated across **ssh_icl_instances** (SSN stream)
- Sticky pass/fail status bit per **ssh_icl_instances** (IJTAG)



- refer to [Sticky Bits](#)

representative_ssh

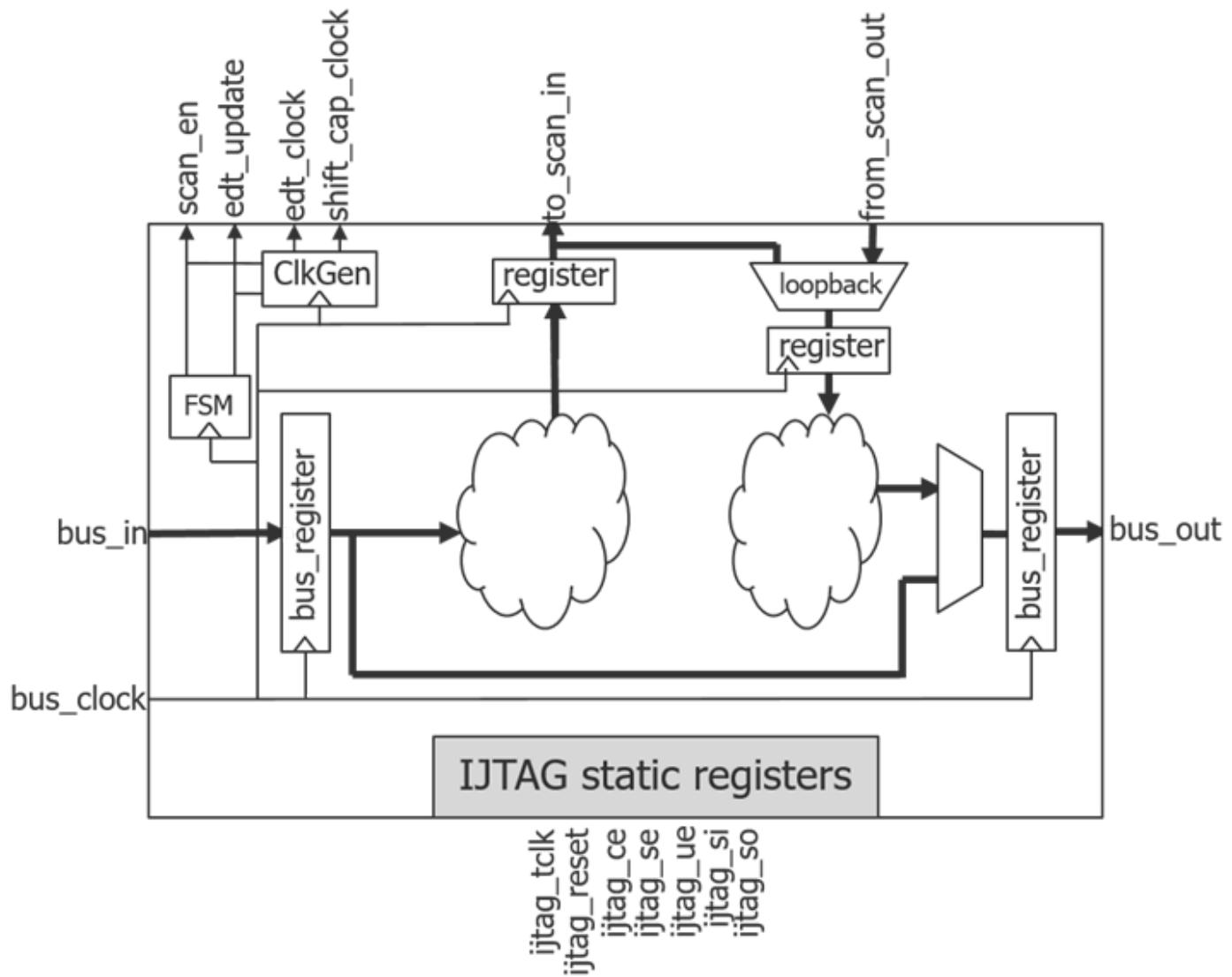
- when multiple identical `ssh_icl_instances` are tested simultaneously as a group, they can share the same scan in data & scan out expected/mask data. In that case, only one copy of the data is present on the ssn bus and one bucket of `time slots` will be allocated. Siemens use the first appeared `ssh_icl_instance` in that group as the `representative ssh`, whose name will appear in the pin/cycle-to-ssh mapping section, and for the following appeared `ssh_icl_instances` they will have an attribute that points to the first ssh, namely `representative ssh`.
- `representative ssh` represent a group of `ssh_icl_instances` that share the same `capture_global_group` ID with it.
- User can't tell if a `ssh_icl_instance` from the mapping section is a `representative_ssh` or not.
- User need to parse the `Active Ssh Section` to get that information.

SMux

- A Multiplexer on the ssn Bus

SSH (ssh)

- Streaming Scan Host



ssh_icl_instance

- an instance of an [ssh](#), accessed via iJTAG with a unique icl address.
- in most cases the term [ssh instance](#) and [icl instance](#) are interchangeable, so it may be referred to as [ssh_icl_instance](#) in the mapping section of a pattern(stil) file.

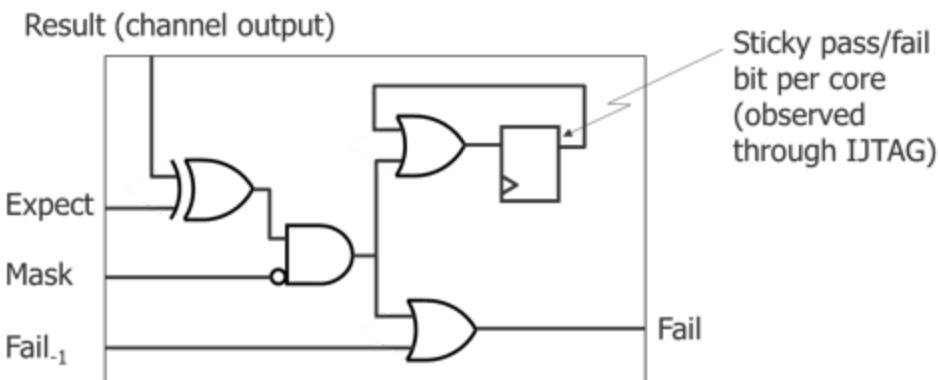
SSN

- Streaming Scan Network

Sticky Bits (sticky_bit)

- A RO bit, if asserted (set to 1), that marks the failure status of a [ssh_icl_instance](#)'s On-Chip compare logic since last reset. (a reset is typically done in [ssn_setup](#) or [ssn_end](#))
- refer to [On-Chip Compare](#)

On chip compare circuitry inside SSH



Tester Compare (TC)

- A ssh takes its scan in data from predictable preallocated `time slot` on the ssn bus and offload the EDT's scan out data on the same `time slot`.

cycle	Repetition 1							Repetition 2							Repetition 3						
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Pin1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1
Pin2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2
Pin3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3
Pin4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4
Pin5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5

- each `time slot` on the ssn bus out will map to a `ssh_icl_instance` (or a group of identical `ssh_icl_instance` that share the same `representative ssh`)

Identifying core instance

```
Ann {*} TESSENT_PRAGMA ssn_mapping -datapath_id 1 -cycle_repetition 7 -cycle_mod 1 -design_port  
"Pin1" -ssh_icl_instance b4.b_rtl1_tessent_ssn_scan_host_1_inst core_instance "b" *)
```

Failure on cycle 15							
	Repetition 3						
cycle	14	15	16	17	18	19	20
Pin1	a1	b4	b2	a2	b5	b3	b1

- Step 1: Calculate Modulus
 - Modulus = cycle % `cycle_repetition`
 - e.g. Cycle 15
 - Modulus = $15 \% 7 = 1$
- Step 2: Lookup core instance from modulus and pin
 - Core instance = b

Pin1 look-up table

Pin	Modulus	Core instance
Pin1	0	a
Pin1	1	b
Pin1	2	b
Pin1	3	a
Pin1	4	b
Pin1	5	b
Pin1	6	b

SIEMENS
TERADYNE

6

- tester will compare the scan out against the expected and any mis-compare will map to the `ssh_icl_instance`, thus one will know which `ssh_icl_instance` failed and which `core_instance` failed.

Pattern Related

ssn_setup pattern

- the pattern for setting up the ssn bus and active ssh instances
- it configs all the `disable_contribution_bit` of active ssh instances
- only the JTAG port is active, the ssn bus is in idle state

ssn_payload pattern

- the pattern that contains all the streaming scan data
- the scan blocks are packed back-to-back without parallel vectors in between
- JTAG and other control pins are in idle state.
- this type of pattern should contain an attribute called `atpg_scan_network`, it's a reference pointing to the [ssn_mapfile.csv](#)

The screenshot shows the PT (Standalone) Pattern Tool interface. The top menu bar includes File, Edit, View, Insert, Debug, Tools, Window, Help, Undo, Redo, Refresh, Auto Refresh, Failing Pins, Align Cycles, Toggle Bookmark, Clear All Bookmarks, Block comments, Update Tester, Open Last Burst, PrePatF, PostPatF, Burst, Halt, First, Previous, Vector, All, Next, Last, Learn... The Properties pane on the left shows Compiler Options like atpg_pntrum_commented (no), atpg_scan_network (yes), compressed (HSDP), digital_inst (no), flexible_scan_pins (no), func_vec_comp (no), import_all_undefineds (no), init_pattern (no), and multiinst (no). The Pattern Explorer pane displays a table of patterns, with the first few rows shown below:

TSet	Command	Label	Vector	Comment
tset_ge_	scan 40 scan_setup	start_label_	0	Pattern:0 Vect
tset_ge_	scan 40 scan_setup		1	1 Pattern:1
tset_ge_	scan 40 scan_setup		2	2 Pattern:2
tset_ge_	scan 40 scan_setup		3	3 Pattern:3
tset_ge_	scan 40 scan_setup		4	4 Pattern:4
tset_ge_	scan 40 scan_setup		5	5 Pattern:5
tset_ge_	scan 40 scan_setup		6	6 Pattern:6
tset_ge_	scan 40 scan_setup		7	7 Pattern:7
tset_ge_	scan 40 scan_setup		8	8 Pattern:8
tset_ge_	scan 40 scan_setup		9	9 Pattern:9
tset_ge_	scan 40 scan_setup		10	10 Pattern:10
tset_ge_	scan 40 scan_setup		11	11 Pattern:11
tset_ge_	scan 40 scan_setup		12	12 Pattern:12
tset_ge_	scan 40 scan_setup		13	13 Pattern:13
tset_ge_	scan 40 scan_setup		14	14 Pattern:14
tset_ge_	scan 40 scan_setup		15	15 Pattern:15
tset_ge_	scan 40 scan_setup		16	16 Pattern:16
tset_ge_	scan 40 scan_setup		17	17 Pattern:17
tset_ge_	scan 40 scan_setup		18	18 Pattern:18
tset_ge_	scan 40 scan_setup		19	19 Pattern:19
tset_ge_	scan 40 scan_setup		20	20 Pattern:20

ssn_end pattern

- the pattern for resetting the ssn bus and active ssh instances
- it also collect the status of all sticky_bits
- only the JTAG port is active, the ssn bus is in idle state

*_SSN.CSV

- a file that is generated along with the ssn_setup pattern or the ssn_end pattern
- contains the extracted active ssh instances information
- the content is subject to change due to the evolution of ssn
- IG-XL currently(as of 11.0) do NOT parse the file, a code lib provided by Factory Apps will read and parse the file.

//ID	Core instance	Icl instance
0	subsystemB/corec_i2	subsystemB/corec_i2/identical_core_rtl_tessent_ssn_scan_host_1_inst
1	subsystemB/corec_i1	subsystemB/corec_i1/identical_core_rtl_tessent_ssn_scan_host_1_inst
2	subsystemB/coreb_i1	subsystemB/coreb_i1/unique_core_rtl_tessent_ssn_scan_host_1_inst
3	subsystemA/corec_i2	subsystemA/corec_i2/identical_core_rtl_tessent_ssn_scan_host_1_inst
4	subsystemA/corec_i1	subsystemA/corec_i1/identical_core_rtl_tessent_ssn_scan_host_1_inst
5	subsystemA/coreb_i1	subsystemA/coreb_i1/unique_core_rtl_tessent_ssn_scan_host_1_inst

Ssh instance	Icl instance	Group ID	Num bits	Contrib pin	Contrib offset	Contrib label	Sticky pin	Sticky offset	Sticky cycle	Sticky label
_core_rtl_tessent_ssn_scan_host_1_inst	subsystemB.corec_i2/identical_core_rtl_tessent_ssn_scan_host_1_inst	2	1	ijtag_tdi	994	disable_contribution0	ijtag_tdo	54	13791	sticky_status0
core_rtl_tessent_ssn_scan_host_1_inst	subsystemB.corec_i1/identical_core_rtl_tessent_ssn_scan_host_1_inst	2	1	ijtag_tdi	809	disable_contribution0	ijtag_tdo	46	13783	sticky_status0
core_rtl_tessent_ssn_scan_host_1_inst	subsystemB.coreb_i1/unique_core_rtl_tessent_ssn_scan_host_1_inst	1	1	ijtag_tdi	624	disable_contribution0	ijtag_tdo	38	13775	sticky_status0
core_rtl_tessent_ssn_scan_host_1_inst	subsystemA.corec_i2/identical_core_rtl_tessent_ssn_scan_host_1_inst	3	1	ijtag_tdi	433	disable_contribution0	ijtag_tdo	24	13761	sticky_status0
core_rtl_tessent_ssn_scan_host_1_inst	subsystemA.corec_i1/identical_core_rtl_tessent_ssn_scan_host_1_inst	3	1	ijtag_tdi	248	disable_contribution0	ijtag_tdo	16	13753	sticky_status0
core_rtl_tessent_ssn_scan_host_1_inst	subsystemA.coreb_i1/unique_core_rtl_tessent_ssn_scan_host_1_inst	1	1	ijtag_tdi	63	disable_contribution0	ijtag_tdo	8	13745	sticky_status0

*_ssn_mapfile.csv

- a file that is generated along with the ssn_payload pattern.
- contains the information of pin/cycle-to-core(ssh) mapping
- the content is subject to change due to the evolution of ssn
- IG-XL will parse and load the file while loading the payload pattern.

//IGXL Pin Name	Cycle Repetition	Cycle Modulo	Core Name
gpo_0	16	0	subsystemB/coreb_i1
gpo_1	16	0	subsystemB/coreb_i1
gpo_2	16	0	subsystemB/coreb_i1
gpo_3	16	0	subsystemB/coreb_i1
gpo_0	16	1	subsystemB/coreb_i1
gpo_1	16	1	subsystemB/coreb_i1
gpo_2	16	1	subsystemB/coreb_i1
gpo_3	16	1	subsystemB/coreb_i1
gpo_0	16	2	subsystemB/corec_i2
gpo_1	16	2	subsystemB/corec_i2
gpo_2	16	2	subsystemB/corec_i2
gpo_3	16	2	subsystemB/corec_i2
gpo_0	16	3	subsystemB/corec_i2

cycle_repetition

- fitting a N bit streaming packet data on a M bit wide bus will result in the mapping pattern to repeat every R cycle, where $R = \text{LCM}(M, N) / M$

- LCM: Least Common Multiple
- it means every R cycles the pin/cycle-to-core(ssh) mapping relation repeats.
 - for each scan out pin, the cycle-to-core(ssh) mapping repeats every R cycles
- we call the R cycle_repetition
- refer to [cycle_modulo](#)

cycle	Repetition 1						Repetition 2						Repetition 3								
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Pin1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1
Pin2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2
Pin3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3
Pin4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4
Pin5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5	b3	b1	a1	b4	b2	a2	b5

cycle_modulo

- refer to [cycle_repetition](#)
- with the mapping being cycling, a given modulo number on a given pin will constantly map to a fixed core
- the mapping table is constructed with 4 columns being:
 - pin
 - cycle_repetition
 - cycle_modulo
 - core/ssh-icl instance

rotation

- fitting a N bit streaming packet data on a M bit wide bus will result in the mapping pattern to repeat every R cycle, where $R = \text{LCM}(M, N) / M$
 - LCM: Least Common Multiple
- sometimes user would patch the streaming data so that N is integer multiple of M, this is called rotation = off
- the most efficient way is to disable patching, which is called rotation = on
- refer to [cycle_repetition](#)

Rotation off

P	BI22	P	BI12	P	BI02
A	BI21	A	BI11	A	BI01
D	BI20	D	BI10	D	BI00
D	AI24	D	AI14	D	AI04
I	AI23	I	AI13	I	AI03
N	AI22	N	AI12	N	AI02
G	AI21	G	AI11	G	AI01
	BI23	AI20	BI13	AI10	BI03
					AI00

- refer to [cycle_modulo](#)

prime_scan_mask (opcode)

- a opCode that is needed for masking core/ssh-icl instance.
- refer to [prime_scan_mask,_prime_no_scan_mask](#)