Acumatica
Virtual DevCon
2020
June 17 & 18

Acuminator and other Developer Tools

Developer productivity, static analysis of code, and solution quality validation

Sergey Nikomarov
System Developer, Platform Team

Evgeny Afanasiev
Technical Account Manager, ISV Team

Agenda

- Quality Validation & Productivity Tools
  - Acuminator Overview & New Features
  - Acumatica Built-in Validations
  - The new Requirement Validation Tool (RVT)
- Integration of Quality Validation Tools into the Development process

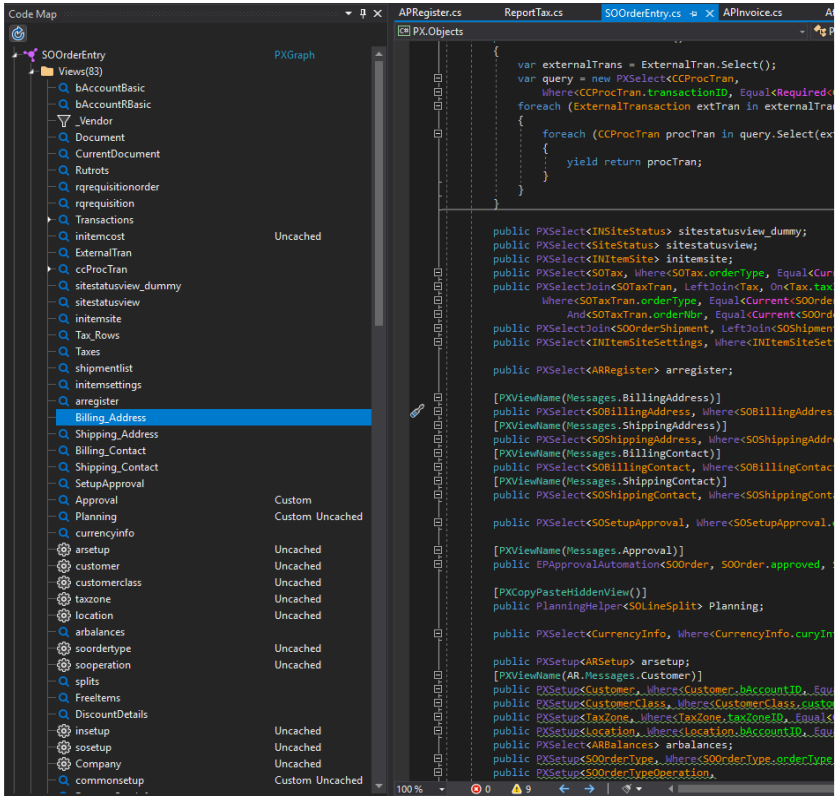
Acuminator

New Features Overview

- Code Map which displays the structure of DACs, Graphs, and their extensions
- Error suppression functionality
- New code analysis diagnostics
- New settings for Acuminator in Visual Studio
- Performance improvements & bugfixes

Acuminator: Code Map

Code Map



Code Map is a tool window in Visual Studio which displays class members in a tree structure.

Code Map is a blend of three technologies: **Roslyn**, **WPF** and **VS SDK.**

Similar generic solutions are present in Visual Studio and other IDEs and productivity tools (Solution Explorer, Object Explorer, etc).

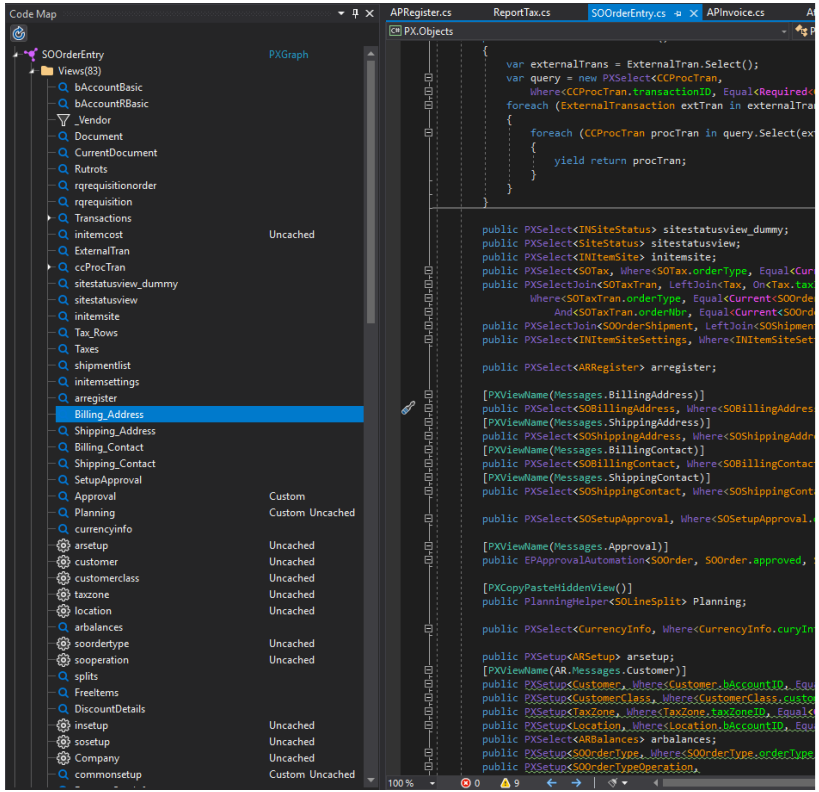
Code Map displays important Graph/DAC members specific for Acumatica Framework.

Code Map supports DACs, Graphs, and their extensions.

Code Map enhances developer productivity.

Key Features

- Quick overview of the code structure
- Grouping of class members together in categories
- Easy navigation to the highlighted code with mouse double click
- Extra information collected by Acuminator
- Ability to sort tree nodes alphabetically or in declaration order
- Synchronization with code changes
- Tooltip with extra information on mouse hover



Use Cases

1. **Code Navigation**
   - Transfer between related code parts during development
2. **Code Investigation**
   - View DAC primary keys.
   - View *Declared Attributes* of a DAC property or *Cache Attached Graph* event
   - View DAC property information: its *type*, whether it is it bound to DB table column or not, etc.
   - Check if a particular action or view has a corresponding delegate
   - Look for a specific event in a Graph
   - Investigate unfamiliar code

Acuminator: Error Suppression

Why Suppress Errors?

There are three common & valid situations when you need to Suppress an Error from the static analyzer:

1. Analyzer is not sophisticated enough, and you know that you can safely violate the static analyzer rule.
2. Code is correct, but the Analyzer gives you a **false positive error**
3. Stuck with a large legacy codebase – But, you would like to check that the new code is correct

These cases result in two different approaches to error suppression:

- First & second cases - **local suppression**
- Third case – **global suppression**

Visual Studio Suppression Functionality

Visual Studio provides its own integrated suppression mechanisms:

- **Pragma directives** like #pragma warning disable PX****
- The `System.Diagnostics.CodeAnalysis.SuppressMessage` attribute
- Special **.ruleset** file
- Suppression in the project settings file.

Acuminator supports all of these, but the available mechanisms cannot fully satisfy our needs.

You cannot conveniently suppress an error for a specified location inside a particular method

Acuminator Error Suppression

Acuminator has error suppression functionality integrated in Visual Studio to suppress its own errors for specified locations:

- Local error suppression with a comment. You can replace **[Justification]** with a justification for suppression

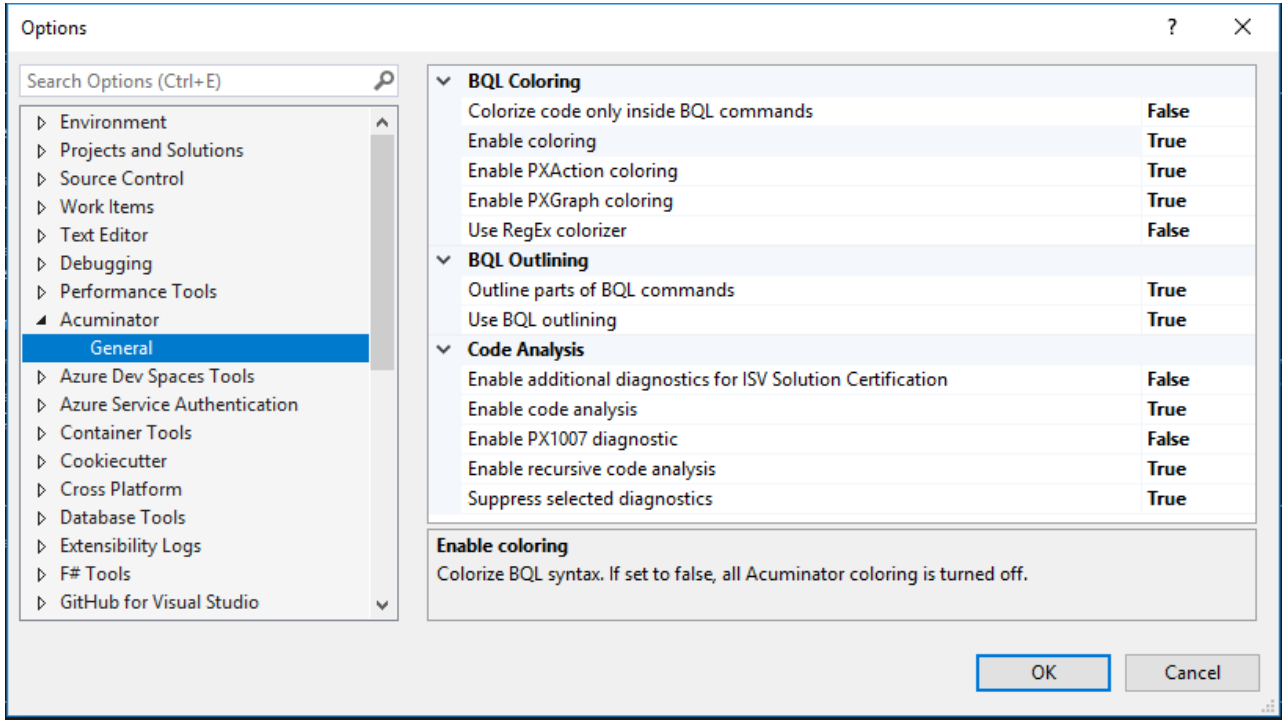
```
// Acuminator disable once PX1092 IncorrectAttributesOnActionHandler [Justification]
public IEnumerable voidInvoice(PXAdapter adapter)
{
    yield break;
}
```

- Global error suppression in the Acuminator suppression file. It is a special file which keeps suppressed errors in the XML format. Its name should be **<Project Name>.acuminator**
- Acuminator supports one suppression file per project and will generate a new suppression file if it does not exist

Acuminator: Settings

Acuminator Settings

Acuminator provides configuration options. To access them in Visual Studio click **Tools** > **Options** > **Acuminator**.



Acuminator Settings

- **Syntax highlight settings**
  - Enable syntax highlighting
  - Highlight only BQL expressions
  - Allow highlighting of Graphs or actions
- **Outlining settings (collapse/expand BQL expressions)**
  - Enable BQL expressions outlining
  - Allow to collapse parts of BQL expression
- **Code analysis settings**
  - Enable code analysis
  - Enable additional diagnostics for ISV solution certification. *This mode will also make analysis stricter by increasing severity of some diagnostics*
  - Suppress selected diagnostic. If set to false, all errors suppressed in the suppression file will be shown

Acuminator: Miscellaneous

Miscellaneous

- **New Diagnostics** — You can find the list of all diagnostics in the Acuminator GitHub repository:

  https://github.com/Acumatica/Acuminator/blob/dev/docs/Summary.md

  *You can also navigate to the diagnostic documentation from the link in the **Error List** window in Visual Studio.*

- **Bugfixes & performance improvements** — If you find a bug in Acuminator you can create an issue on GitHub:

  https://github.com/Acumatica/Acuminator/issues

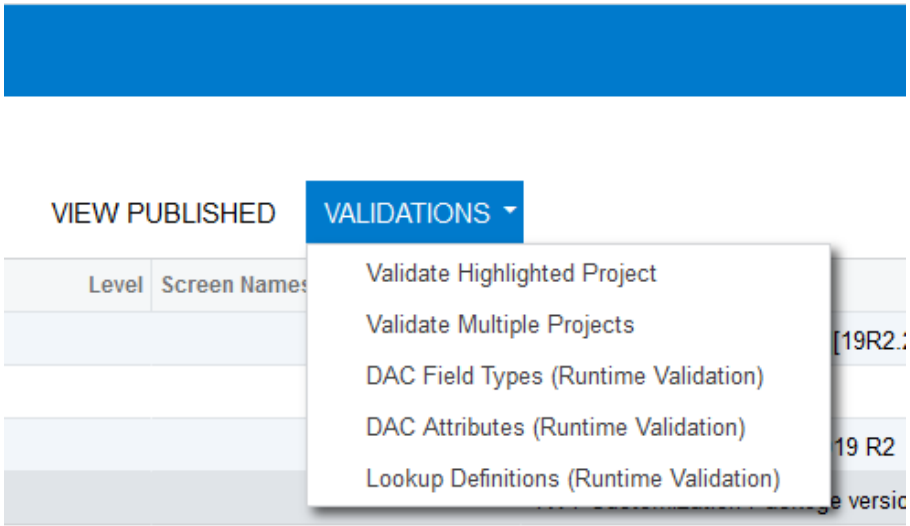
- Acuminator can be installed as a NuGet package **Acuminator.Analyzers**.

  This makes Acuminator checks mandatory for the whole team. However, you need to install VSIX to use the Code Map, BQL Formatter and suppression in the suppression file.

Acumatica Built-in Validations

Acumatica Built-In Validations

- Project Validation — The validation of the single highlighted project or multiple projects content
- DAC Field Types Validation
- DAC Attributes Validation
- Lookup Definitions Validation



Acumatica Built-In Validations Tips

- Project Validation is not the so-called process that triggered prior to publishing
- Project Validation should be performed before publishing
- Runtime Validations will include the solution related results ONLY after its publishing and will always include validation of the entire Acumatica instance.
- Lookup Definitions Validation results are pointing on the issue with attributes that decorates selector fields *(often due to the application of Segmented Keys)*

**Some more information about you might find in official help here: Validation of a Customization Project*

Requirement Validation Tool (RVT)

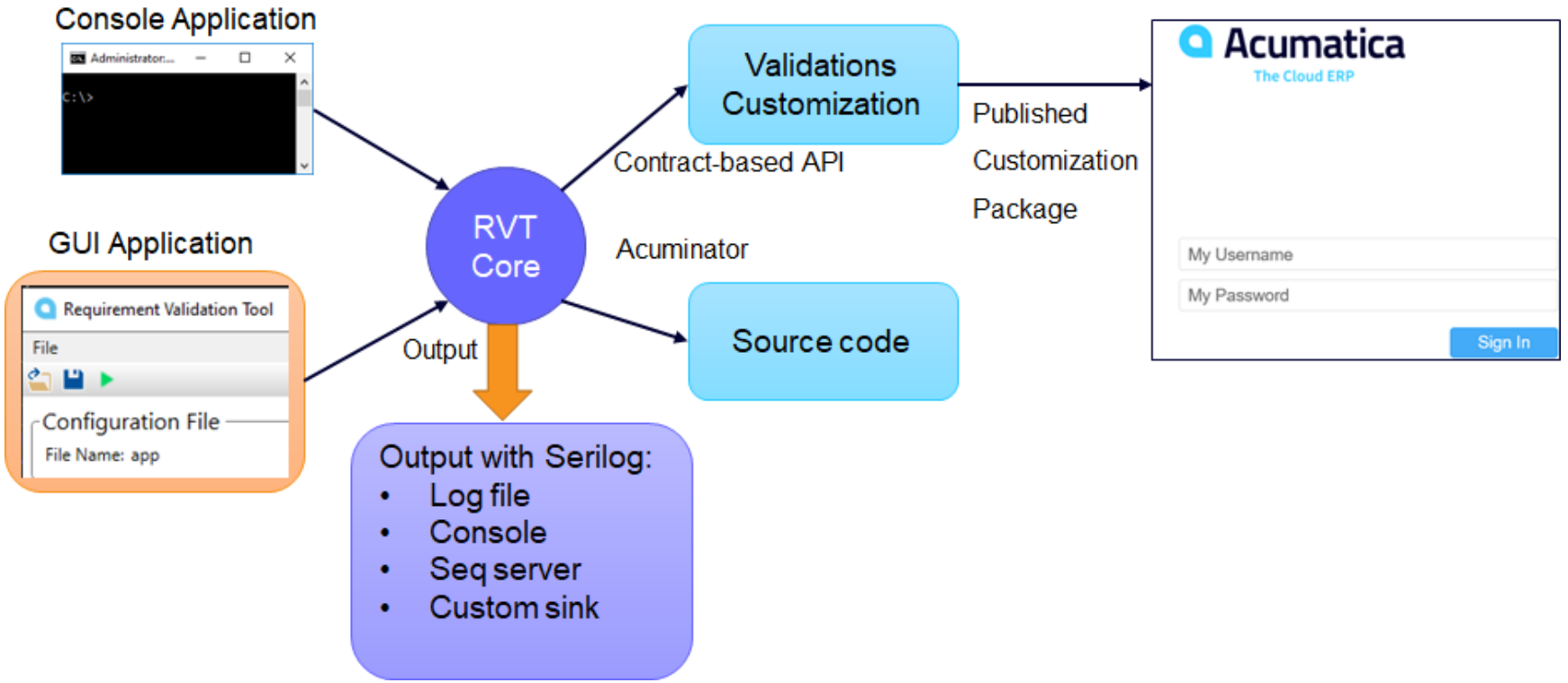
RVT: Requirement Validation Tool

RVT is a standalone configurable tool that allows to perform various validations of embedded and composite ISV solutions in the context of the Acumatica ERP local instance.

The following validations are currently available:

- Acuminator Validation
- Screen Schema Validation
- Translation Validation

RVT: Key Parts



RVT: Acuminator Validation

Project validation with Acuminator's analyzer — designed to work within *Continuous Integration* and has following features/requirements:

- Requires source code in form of Visual Studio Project (*.**csproj)**
- Project should have all the dependencies linked and compilable
- RVT Acuminator Validation is enabled to generate suppression files and use them for further iteration

RVT: Screen Schema Validation

Validates the `screenInfo` of Acumatica screens, customized and new screens.

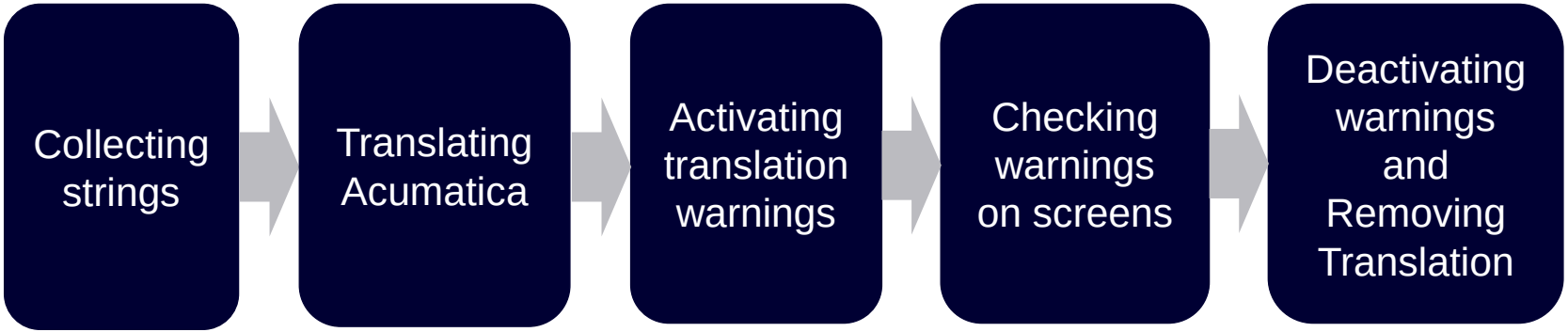
The `screenInfo` is used in all Acumatica APIs: Contract-based, Screen-based, Import/Export, Mobile.

Screen Schema Validation checks the following:

- `screenInfo` is available and is generated without errors
- PrimaryView is bound to single container control
- BLC (Graph) type is used by a single page

RVT: Translation Validation

Validates that Acumatica can be translated using the built-in localization mechanism.



**Unbound strings are not covered by this validation and could be checked by Acuminator instead.*

RVT: References

Download is available here:

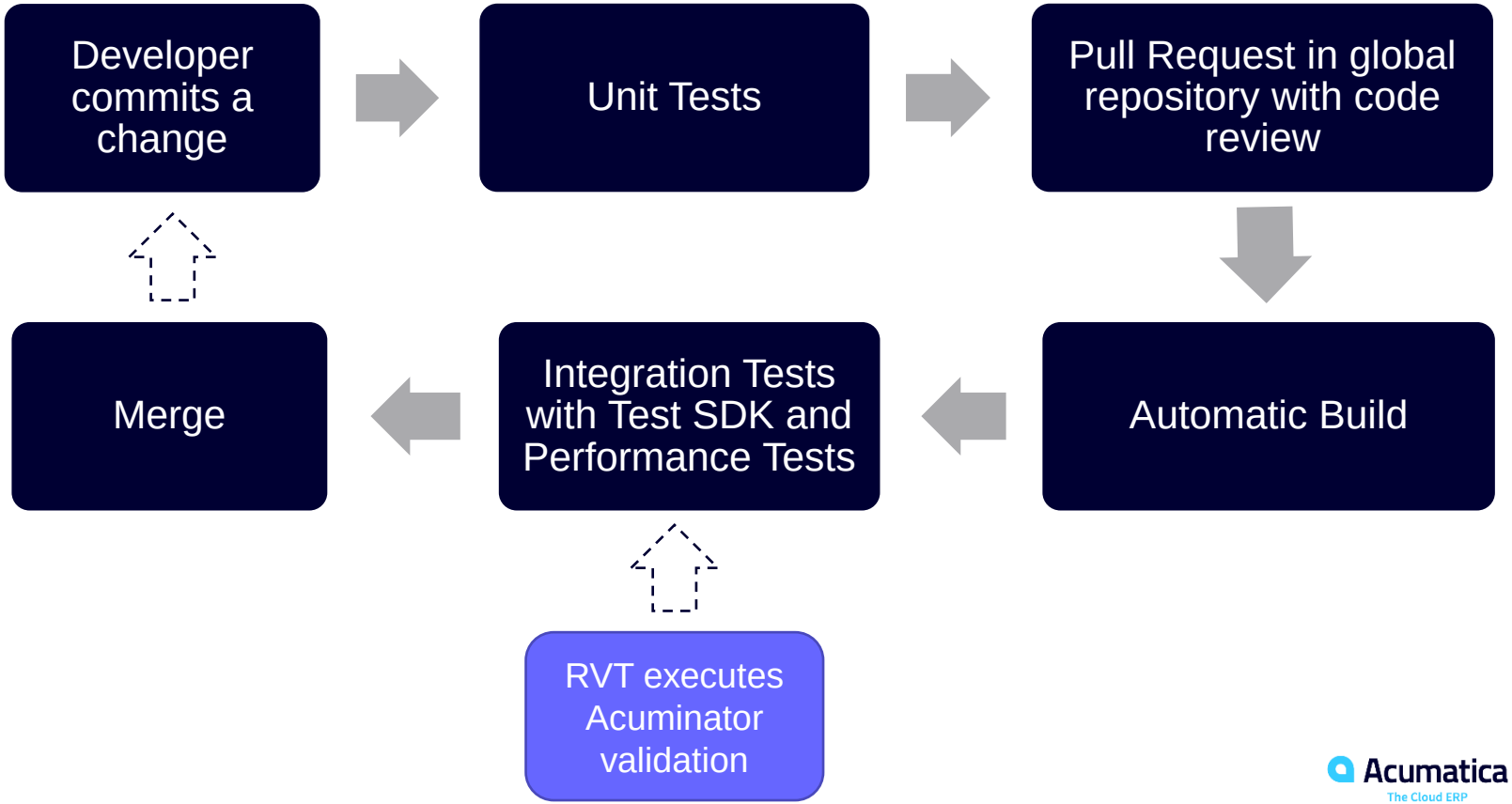
Requirements Validation Tool v. 2.4

Documentation

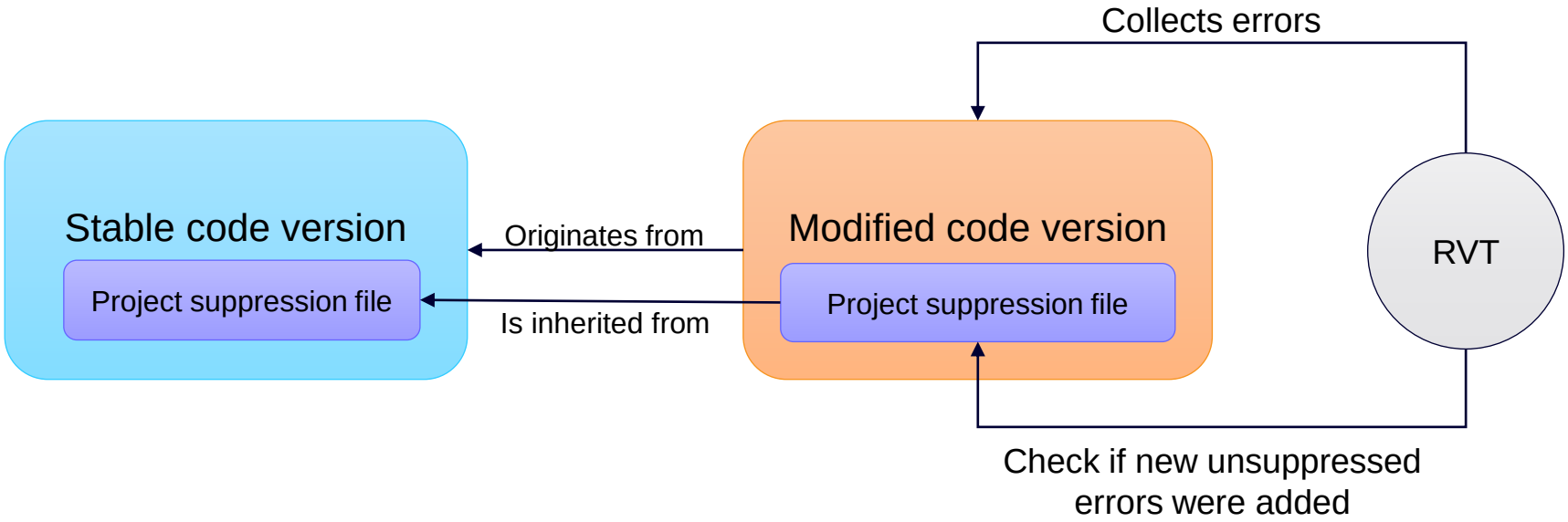
Please give us your feedback on certification@acumatica.com

RVT Integration into Development Process

Acumatica Development Process and RVT



How Acuminator Validation by RVT Works



Integration with RVT for Acuminator static analysis

RVT can be simply integrated into your automated testing with a script:

- RVT has a console runner **RequirementsValidation.exe**
- The runner reads settings from a standard configuration file **RequirementsValidation.exe.config**
- To perform RVT Acuminator check you need to:
    1. Make sure that the environment is configured on the test machine
    2. Place the code to be validated in a prepared location used by test
    3. Build the prepared solution or restore NuGet packages and place all external DLLs to their expected locations
    4. Make sure that RVT is present on the test machine and configure it
    5. Run RVT and check the process exit code. If it equals 0, then the validation is passed.
    6. Process the results

Configuration Example

```
var rvtConfigPath = <provide path to config file here>;
var rvtConfig = XDocument.Load(rvtConfigPath);
XElement appSettings = rvtConfig.Root.Element("appSettings");

SetAcuminatorSetting("AcuminatorRequirement_Validate", true);
SetAcuminatorSetting("AcuminatorRequirement_ProjectPath", <path to project>);
SetAcuminatorSetting("AcuminatorRequirement_IsvSpecificAnalyzersEnabled", true);
SetAcuminatorSetting("AcuminatorRequirement_PX1007DiagnosticIsEnabled", false);
SetAcuminatorSetting("AcuminatorRequirement_GenerateSuppressionBase", true);
SetAcuminatorSetting("AcuminatorRequirement_ValidateSuppressionBaseDiff", true);
rvtConfig.Save(rvtConfigPath);

//---------------Local Function---------------
void SetAcuminatorSetting(string name, object value) =>
    appSettings.Elements("add")
               .Single(e => e.Attribute("key").Value == name)
               .Attribute("value")
               .SetValue(value);
```

RVT integration with the Test SDK

Using RVT with the Test SDK

You can write an integration test using the Test SDK which will run RVT validation.

Create a new class derived from the **Check** class of the Test SDK and override the **Execute** method.
Then you can prepare environment as necessary:

```
public class AcuminatorValidation : Check
{
        public override void Execute()
        {
           //You can prepare environment from the test if needed
           InstallMSBuildIfNotInstalled();
           string dotnetPath = InstallDotNetRunTimeIfNotInstalled();
           InstallGitIfNotInstalled();

           //prepare code repository
           string codeRepoPath = PrepareCodeRepository(dotnetPath);
           string projectPath = GetValidatedProject(codeRepoPath);
           string rvtPath = ConfigureRVT(projectPath);           //see previous example

           RunAcuminatorValidation(rvtPath, dotnetPath);
           //You can optionally process results after the validation
        }
}
```

Code Repository Preparation

```
public override void PrepareCodeRepository(string dotnetPath, bool buildSolution = true)
{
    string codeRepoPath = CloneCodeRepository();
    string solutionPath = GetSolutionPath(codeRepoPath);

    // You can either build solution or just restore nuget packages and copy external DLLs
    if (buildSolution)
    {
        BuildSolution(solutionPath);
    }
    else
    {
       //restore nuget packages by running nuget CLI (see next slide)
       RestoreNugetPackages(solutionPath, dotnetPath);

       //Copy external DLL dependencies to prevent compilation errors
       CopyExternalDLLsToExpectedLocations(solutionPath);
    }

    return codeRepoPath;
}
```

Restore Nuget Packages

```
public override void RestoreNugetPackages(string solutionPath, string dotnetPath)
{
    const string nugetPath = <path to nuget command line tool>;

    //restore nuget packages by running nuget CLI in a process
    using (var process = new Process()) {
        process.StartInfo =
                new ProcessStartInfo(nugetPath, "restore " + solutionPath)
                {
                        UseShellExecute = false
                };

        process.StartInfo.Environment["MSBuildSDKsPath"] = dotnetPath;
        process.Start();
        process.WaitForExit();

        if (process.ExitCode != 0)
                throw new AutotestException("Failed to restore nuget packages");
     }
}
```

Run RVT Validation

```
public override void RunAcuminatorValidation(string rvtPath, string dotnetPath) {
    using (var process = new Process())
    {
        process.StartInfo =
                new ProcessStartInfo(rvtPath)
                {
                    WorkingDirectory = Path.GetDirectoryName(rvtPath),
                    UseShellExecute = false,
                    RedirectStandardOutput = true
                };
                process.StartInfo.Environment["MSBuildSDKsPath"] = dotnetPath;

                process.Start();
                var output = process.StandardOutput.ReadToEnd();
                process.WaitForExit();

                if (process.ExitCode != 0 || output.Contains("Error"))
                    Log.Error(output);
                else
                    Log.Information(output);
    }
}
```

Sumary

Tools that simplify the development with Acumatica & ensure the quality of the product:

- **Acuminator** – new features to boost programmer productivity and help with code analysis in Visual Studio.

  https://github.com/Acumatica/Acuminator

- **Project Validations** – a set of powerful run-time validations embedded in the Acumatica site: https://help-2020r1.acumatica.com/Help?ScreenId=ShowWiki&pageid=f048f674-63dc-4ad4-b2ba-4b7ec43cfc95

- **Requirement Validation Tool** — a new instrument to perform the validation of the entire solution. It can be integrated into your automated testing.
  http://acumatica-builds.s3.amazonaws.com/index.html?prefix=builds/tools/RequirementsValidation/2.4.0.0/

Thanks for hard work and help

Big thanks to the team of contributors:

- Vladimir Panchenko
- Vyacheslav Yakutenko
- Dmitry Naumov
- Andrey Budaev
- Kseniya Popova
- Ekaterina Androsova

Questions?

Acumatica
The Cloud ERP

Sergey Nikomarov

snikomarov@acumatica.com

Evgeny Afanasiev

evgeny.afanasiev@acumatica.com