inrule

InRule Developer Guide

Help Documentation for the InRule Decision Platform

Date of Publication: 3/22/2021

---- This page intentionally left blank ----

InRule® Developer Help

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Published: March 2021 in Chicago, IL

Table of Contents

Part I

InRule SDK Developer's Guide

1 InRule SDK Developer's Guide

Welcome to the InRule SDK Developer's Guide

InRule® can be developed with and deployed in a variety of configurations in order to meet the your business needs while complying with your enterprise standards and practices. To facilitate this, InRule provides an extensive .NET object model to allow tight integration with your end business applications.

Nearly any type of application or process can benefit from integrating with the InRule rules engine and catalog. Whether web-based, smart client, or a services oriented architecture, InRule can provide centralized decision-making to drive interfaces and manage enterprise-wide computations.

The following sections describe how to integrate InRule into your application:

- Migration Considerations
- irSDK® Object Model
- Application Integration with InRule
- Implementation Guide
- Source Code Examples
  - Runtime API
  - Authoring API
  - Catalog API
- InRule Samples
- Regression Testing
- Rule Tracing

Part II

Migration Considerations

2 Migration Considerations

Overview

As of InRule version 4.5.0 and later, several changes to rule applications, file formats, configuration settings, and the InRule SDK were introduced, including changes to some of the core classes and methods to execute rules. Please refer to the following resources for guidance on how to change your applications to be compatible with this new version:

- This “Migration Consideration” section
- Transition Guide, which can be downloaded from the InRule Support website using the following link: https://support.inrule.com/cs/media/p/968.aspx
- Runtime API Examples in this help file
- InRule Samples on GitHub

User Assemblies

As of version 4.5, InRule no longer supports the placement of user assemblies in the irAuthor® folder. You should instead copy user assemblies to the `irAuthor\EndPointAssemblies` folder. The standard load sequence for user assemblies is:

- The Global Assembly Cache (GAC)
- CurrentAppDomain.BaseDirectory
- Configured EndPointAssemblies directory

Rule Application Changes

The following rule application functions have changed in version 4.5:

- **Fire Event Action.** The Fire Event Action has been replaced with the Fire Notification action. Existing rule applications will be upgraded automatically.
- **Web Service Endpoint.** Existing rule applications which contain Web Service endpoints will be automatically upgraded when opened. However, if the web service URI stored in the rule application is invalid at the time of the upgrade, any Execute Web Service Action mapping entries will be lost. If the rule application is opened from the file system, the irAuthor user will be warned about the upgrade; if the rule application is stored in the catalog, this warning will be presented during the catalog upgrade operation.
- **XML Schema Endpoint.** Existing rule applications which contain XML Schema endpoints will require an authoring user to reload the XSD (unless embedded), then apply to schema (even if embedded).
- **Database Schema/XSD DataSet Schema.** Existing rule applications using a Database Connection Schema or an XSD Schema where the XSD is a DataSet XSD will work in basic scenarios, such as modify XML-loaded state and get XML. However, automatic constraint and identity-field assignment on Add Collection Member will not work. Please contact InRule for assistance in implementing a custom state provider to directly update a DataSet instance.
- **Default Value with Object-Bound State.** When using a .NET assembly endpoint, default values authored on the rule application for any bound entity fields are ignored, unless runtime binding is explicitly disabled via the irSDK.
- **Default Value with non-Object-Bound State.** When not using a .NET assembly endpoint, the default value behavior is similar to version 4.1, with one important distinction. In version 4.5, InRule applies the default value only when an entity is first created, while version 4.1 applies the default value any time the field is null (including if an action sets it to null during rule execution).
- **Aggregate Function Syntax.** InRule version 4.1 allowed a user to specify a syntax such as `Sum(Collection1.Collection2, Collection2Field1, "filterval")`. The supported

syntax in version 4.5 is `Sum(Collection2, Collection2Field1, "filterval")`. For example, `Sum(Vehicles, Premium, Year=1999)`.

Configuration Changes

The following configuration file settings have changed.

InRule Repository Configuration

Sub Section	Setting Name	Changes
assemblyEndPoint	shadowCopyDirectory	Removed
	shadowCopyEndPointAssemblies	Removed
licensing	cryptographyRequired	Removed

InRule Runtime Configuration

Sub Section	Setting Name	Changes
dbCommand	dbCommandTimeout	Removed
locale	culture	Removed
ruleEngine	emptyNodeForValueTypesOutputFormat	Removed
	nullZeroEquality	Removed
	ruleTimingEnabled	Removed
webServiceConnections	disableBindingOverrides	Removed
workingMemory	workingMemoryCacheCleanUpInterval	Removed
	workingMemoryCacheTimeout	Removed

InRule Authoring/Client Configuration

Sub Section	Setting Name	Changes
catalogClient	maxItemsInObjectGraph	Removed
ruleEngineClient	ruleExecutionTimeout	Removed
	runawayCycleCount	Removed
	useRuleEngineServiceForIrVerify	Removed

	useRuleEngineServiceRepositoryService	Removed
tracing	bufferLogWrites	Removed
	bufferSize	Removed
	logActivities	Removed
userInterface	helpFileDirectoryPath	Removed
	MRUFileNamesMaxLength	Removed
	ShowDebugForEntityStateViewerinIrVerify	Removed
	udfEditorAutocompletionEnabled	Removed

Test Scenario File Changes

Test Scenario files are now stored in a proprietary binary format, rather than the XML format of previous InRule versions. Older Test Scenario files will not work with InRule version 4.5. If you require the use of prior release Test Scenario files, download the Test Scenario file upgrade program from the InRule support site, and follow the instructions on the site.

User Defined Function Changes

Refer to the InRule Authoring Guide for changes to User Defined Functions.

irSDK Changes

The following assemblies have been removed:

- inrule.common.xmlserializers.dll
- inrule.runtime.xmlserializers.dll
- inrule.scripting.dll

The following namespaces have been removed from irSDK:

- InRule.Runtime.Messages
- InRule.Runtime.Metadata
- InRule.Runtime.Statistics

Changes in the Runtime API (InRule.Runtime)

Class	Member	Changes
CachedRuleApp		Removed
Collection	AddCollectionMember	Add()
	AppendCollectionMember	Add(object)
	CollectionDef	GetDef()
CollectionMember	GetEntity	Value.ToEntity()
CompilerErrorInfo		CompilerError

ElementValue	ToDouble	Removed
	ToElementIdentifier	Removed
	ToEntityIdentifier	Removed
	ToInt16	Removed; use ToInt32()
	ToString(bool)	Removed; use ToString()
Entity	ChildRuleElements	RuleSets
	CollectionChanged	Removed
	EntityDef	GetDef()
	EntityId	ElementId
	EntityStateLoaded	Removed
	ExecuteRuleSet(string, RuleSetParameter[])	ExecuteRuleSet(string, object[])
	GetCollection	Collections[string]
	GetField	Fields[string]
	GetField(XPathExpression)	Removed

	State.GetXml	WriteXml(XmlWriter)
		for example:
		`StringBuilder sb = new StringBuilder();`
		`XmlWriter writer = XmlWriter.Create(sb);`
		`entity.WriteXml(writer);Return sb.ToString();`

	State.Load	ReadXml(string)

	State.LoadXml	ReadXml(XmlReader)
		for example:
		`entity.ReadXml(XmlReader.Create(new StringReader(string xmlString)));`

	State.Save	WriteXml(string)
	ValidityChanged	Removed
	ValueChanged	Removed
EntityIdentifier		Removed; use ElementId, which is a string
Field	FieldDef	GetDef()
	GetValueList()	AssociatedValueList
	SetValue	Value
FileSystemRuleApp		FileSystemRuleApplicationReference
FixedRevisionRuleApp		Removed

InMemoryRuleApp		InMemoryRuleApplicationReference
InProcessConnection		Removed
ListItemValue		ValueListItem
ObjectEntityState		Removed; to bind an Entity, use RuleSession.CreateEntity(string, object)
RepositoryRuleApp		CatalogRuleApplicationReference
RuleApp		RuleApplicationReference
RuleCompileErrorsException		CompileException (in InRule.Repository)
RuleElement	RuleElementDef	GetDef()
RuleRuntimeErrorsException		RuntimeException
RuleServiceConnection		Removed
RuleSession	ActivateRuleSets(string[])	ActivateRuleSets(string)
	ActivateRuleSetsByCatagory(string[])	ActivateRuleSetsByCategory(string)
	AggExecStats	Removed; use Statistics or assorted new members in the RuleSession class
	AggExecStats.AggExecStatInfo.GetHtml()	LastRuleExecutionLog.GetHtml()
	CreateRuleSet	CreateIndependentRuleSet
	DataElementOverrides	Overrides.Override*()
	DeactivateRuleSets(string[])	DeactivateRuleSets(string)
	EventFired	Removed; the "fire event" action is no longer supported
	LoadState	Syntax is unchanged, but the RuleSession state file is a proprietary format instead of XML
	RuleApplicatonDefInfo	GetRuleApplicationDef()
	RuleApplicationInfo	Removed
	State.GetActiveNotifications	GetNotifications()
	State.GetActiveValidations	GetValidations()

RuleSessionSettings	RuleExecutionSettings	Removed; use assorted new members in the RuleSessionSettings class
	RuleExecutionSettings.CurrentDateOverride	Now
	RuleExecutionSettings.EnableRuleExecutionTracing	LogOptions; uses the EngineLogOptions enum
	RuleExecutionSettings.ExecutionTimeoutOverride	ExecutionTimeout
	RuleExecutionSettings.IncludeAttributeTables	Removed
	RuleExecutionSettings.IncludeDescriptiveDetail	Removed
	RuleExecutionSettings.IncludeRules	Removed
	RuleExecutionSettings.MaxEvaluationCyclesOverride	MaxCycleCount
	RuleExecutionSettings.ReturnDetailStatisticsInfo	LogOptions; uses the EngineLogOptions enum
	MaxDegreeOfParallelism	Use MaxExecutionCores
RuleSet	ChildRuleElements	RuleElements
	RuleSetDef	(RuleSetDef)GetDef()
RuleSetParameter		Removed
TransactionMessage		LogMessage
Validation	InvalidMessageText	Message
WebServiceConnection		Removed

Changes in the Catalog API (InRule.Repository)

The Evaluation Network (EvalNetwork) and related classes have been replaced by the DefUsageNetwork. See Determine FieldDef Dependencies for a sample of how to use the DefUsageNetwork.

Additional Considerations (from 3.x)

If you are upgrading from InRule version 3, then please note these additional considerations:

- InRule Version 4 requires the Microsoft .NET 4.0 runtime.
- Licensing for InRule version 4 will require a different serial number than the number used to activate version 3. Please contact your sales representative to obtain an InRule version 4 serial number. The InRule.lic file is no longer supported, and licenses must be specifically registered on each machine on which InRule is used.

- WinForm authoring controls have been removed from irSDK. These controls have been replaced with WPF 4.0 controls that are publicly available.
- ASP.NET authoring controls have been removed from irSDK. These controls have been replaced with Silverlight 4.0 controls that are publicly available.

Assembly changes

Assembly	Change	Change description
InRule.Authoring.Editors	Added	This DLL is required to consume InRule WPF controls in a custom rule authoring application.
InRule.Authoring.Windows.dll	Added	This DLL is may be required to consume some InRule WPF authoring functionality in a custom rule authoring application. This DLL may be copied as a dependency if InRule. Authoring.Editors is referenced
InRule.Authoring.dll	Added	This DLL is may be required to consume some InRule WPF authoring functionality in a custom rule authoring application. This DLL may be copied as a dependency if InRule. Authoring.Editors is referenced
InRule.Authoring.UI.dll	Removed	This DLL, along with the Windows Forms controls it contained in InRule 3.x are no longer supported.

Part III

irSDK Object Model

3 irSDK Object Model

InRule's irSDK contains a rich object model providing all of the properties, methods, and events required to manage most rule-enabled scenarios. Below are the core assemblies and namespaces that comprise irSDK. The assemblies are installed in <InRule installation directory>\irSDK\bin by default.

The namespaces contained in irSDK fall into two logical groups:

- Runtime API - runtime execution namespaces for requesting rules execution and processing results
- Authoring API - rule authoring namespaces for creating rules in code and embedding authoring controls

Runtime Execution Namespaces

InRule.Runtime

- Used to execute rule applications
- Typically the primary namespace used by developers to call the rules engine
- Assembly: `InRule.Runtime.dll`

InRule.Runtime.Testing

- irVerify® integration and execution statistics

Rule Authoring Namespaces

InRule.Repository

- Used to create and maintain rule applications
- Every necessary rule element is available to create rule applications in code
- Both irAuthor uses this namespace exclusively
- Assembly: `InRule.Repository.dll`

InRule.Authoring.Editors, InRule.Authoring.BusinessLanguage

- Classes and methods to embed authoring controls in a host application
- Assemblies: `InRule.Authoring.Editors.dll`, `InRule.Authoring.BusinessLanguage.dll`

InRule.Security

- Set of classes and namespaces to assign rights to element types with the rule application
- Assembly: `InRule.Repository.dll`

Part IV

Application Integration with InRule

4 Application Integration with InRule

The InRule Rules Engine can be integrated into end applications in many different ways. Typical implementations range from calling the rules engine as a simple calculation engine to driving dynamic surveys incorporating rules and metadata to influence UI display, validation, and navigation.

Application Integration Topics

- InRule Product Architecture
- Rule Execution Process Flow
- irSDK Assembly Information
- Using InRule NuGet Packages
- Configuration Files
- Performance Logging and Monitoring
- irServer® - Rule Execution Service
- irAdapter for BizTalk Server
- InRule Activity for Windows Workflow
- .NET Assembly State Refresh Options
- InRule Culture Settings
- Interacting with Non .NET Platforms
- Embedded Authoring Control
- Customizing irAuthor with Extensions
- License Activation Utility
- InRule Temp Files
- irSDK for .NET core

Core Runtime Objects used to call the Rules Engine

Refer to the source code example: Basic Example of Calling the Rule Engine to see how to use these objects in code.

RuleSession

- Session object that manages all of the rules engine request directives and execution results

RuleApplicationReference

- Instance of a runtime rule application
- Derived Classes: FileSystemRuleApplicationReference, CatalogRuleApplicationReference, InMemoryRuleApplicationReference

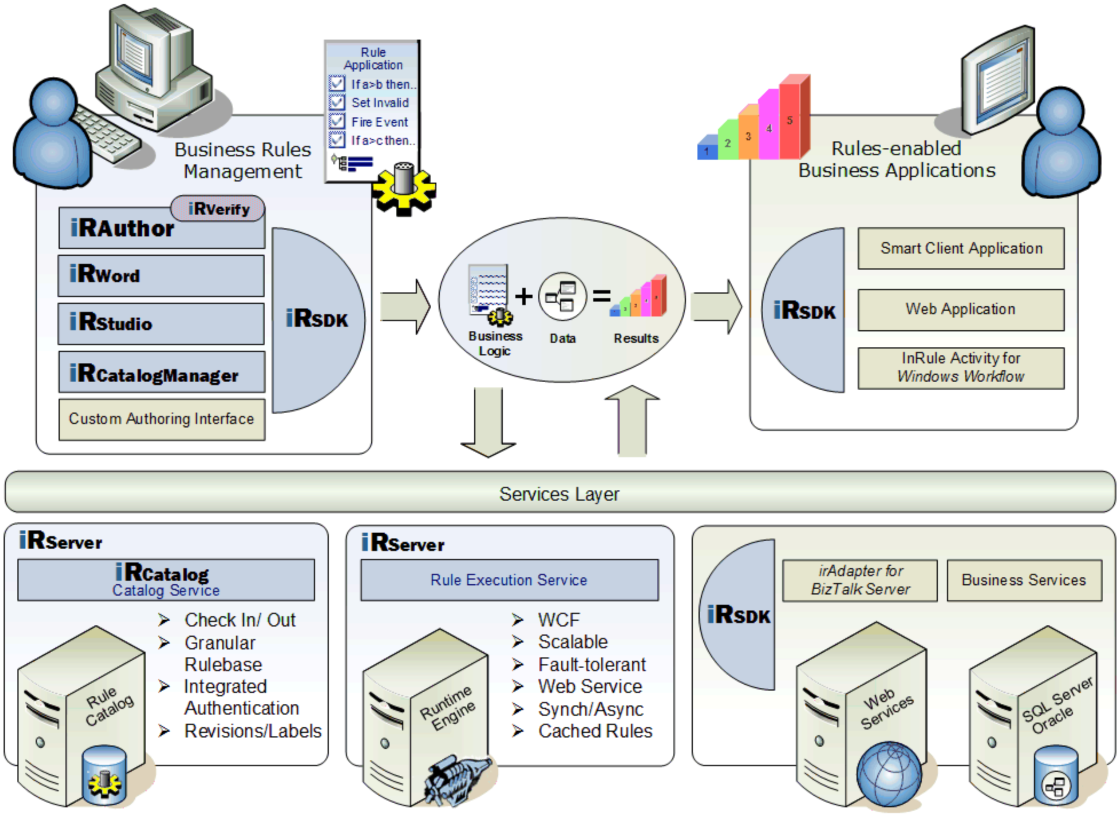
Entity

- Holds data for the rules engine
- State may be saved and loaded as needed

4.1 InRule Product Architecture

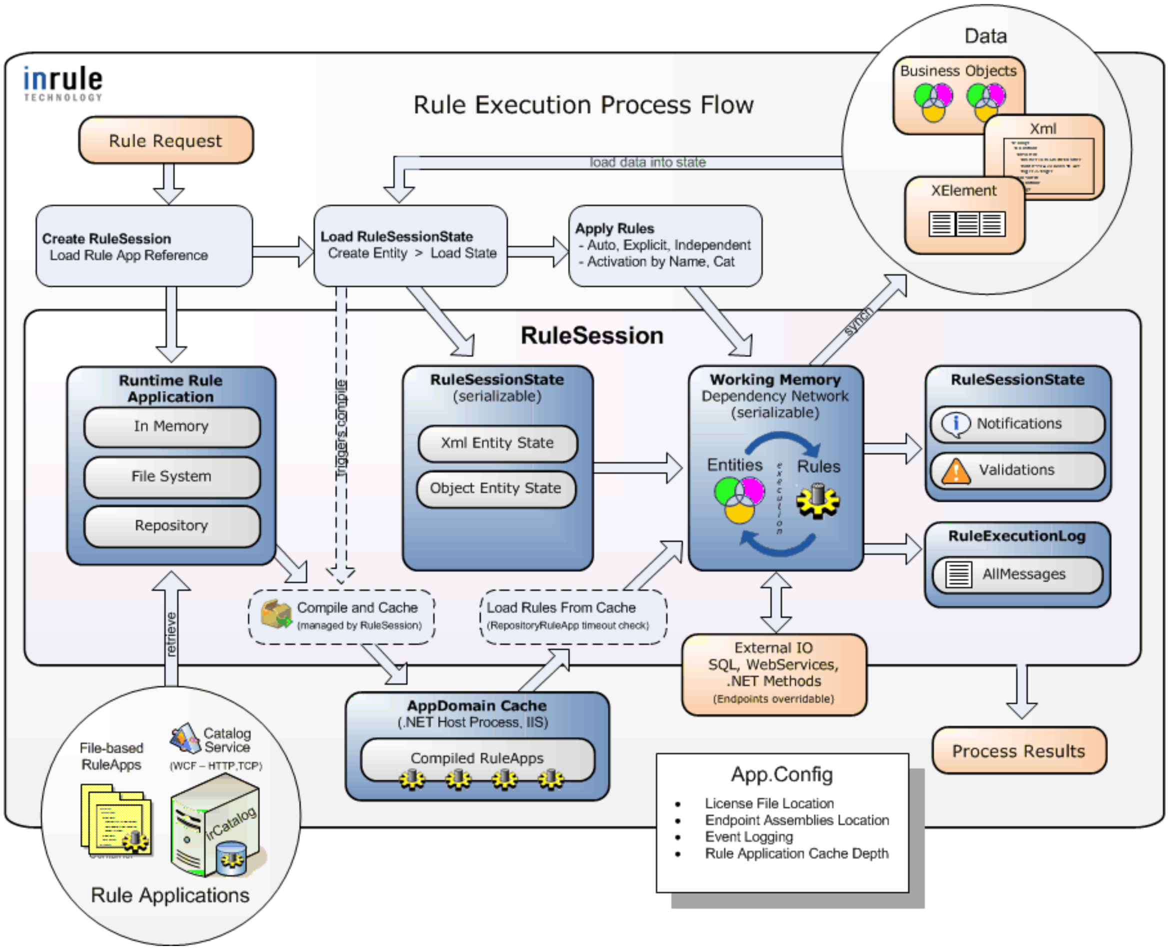
InRule's product architecture consists of:

- End User Tools: irAuthor, irVerify, irWord™, irCatalogManager (deprecated)
- Services: irCatalog® Service, irServer Rule Execution Service
- Developer Tools: irSDK



4.2 Rule Execution Process Flow

InRule's Rule Execution Process Flow is typified by the following diagram:



4.3 irSDK Assembly Information

InRule's SDK contains a number of assemblies, combinations of which are needed for different authoring and runtime scenarios.

The tables below cover the most common scenarios.

Scenario	Assemblies
Custom authoring application (WPF or WinForms)	• ActiproSoftware.Shared.Wpf • ActiproSoftware.SyntaxEditor.Wpf • InRule.Authoring • InRule.Authoring.Authentication • InRule.Authoring.Editors • InRule.Common • InRule.Repository • InRule.Runtime.Testing (only if irVerify is used)
irAuthor extension	• InRule.Authoring

	• InRule.Authoring.Authentication • InRule.Authoring.Editors • InRule.Authoring.Windows • InRule.Common • InRule.Repository
Rule execution	• InRule.Common • InRule.Repository • InRule.Runtime
Windows Workflow Foundation	• InRule.Activities • InRule.Common • InRule.Repository • InRule.Runtime

4.4 Using InRule NuGet Packages

When you install irSDK, NuGet packages for the SDK assemblies will also be placed in a "NuGetPackages" sub-folder.

Packages

Name	Description
InRule.Activities	Support for using InRule with Windows Workflow Foundation.
InRule.Authoring.Core	Core Authoring SDK assemblies used for creating irAuthor extensions. This package is also required for creating custom authoring applications.
InRule.Authoring.SDK	Authoring SDK assemblies used for creating custom authoring applications. Includes assemblies needed at runtime when executing outside irAuthor. irAuthor extension developers do not need this package and should use the InRule.irAuthor.SDK package instead.
InRule.irAuthor.SDK	irSDK Authoring assemblies used for irAuthor extensions. This is a metapackage with dependencies on other packages needed for extension creators.
InRule.Common	Common irSDK assembly used for integration with InRule®.
InRule.Repository	Common Repository irSDK assemblies used for authoring and/or runtime integration with InRule®.
InRule.Runtime	Runtime integration with the InRule® Rule Engine.
InRule.Runtime.Salesforce	Assembly to facilitate mapping between InRule Entities and Salesforce® Platform objects.

Licensing

The packages are licensed under the standard InRule End User License Agreement and the packages metadata includes a link to this license. Installing the packages via most NuGet clients will require you to explicitly accept the license.

When executing applications that consume these assemblies, the associated InRule license must be activated.

Package Dependencies

In most cases, packages will depend on other packages based on the assembly dependencies. For example, installing the InRule.Authoring.Core package will install the Authoring SDK assemblies, and it depends on the InRule.Repository package in order to also install InRule.Repository.dll. The InRule.Repository package depends on the InRule.Common package, which contains InRule.Common.dll. It is generally only necessary to install the specific package you need for your development scenario (InRule.irAuthor.SDK or InRule.Runtime, for example) and allow the dependencies to install the remaining packages, if any.

Dependencies are set to specific versions, so if you upgrade a single package, all of the InRule packages will also upgrade at the same time in order to meet the dependency requirements. Typically, your NuGet client will handle this for you.

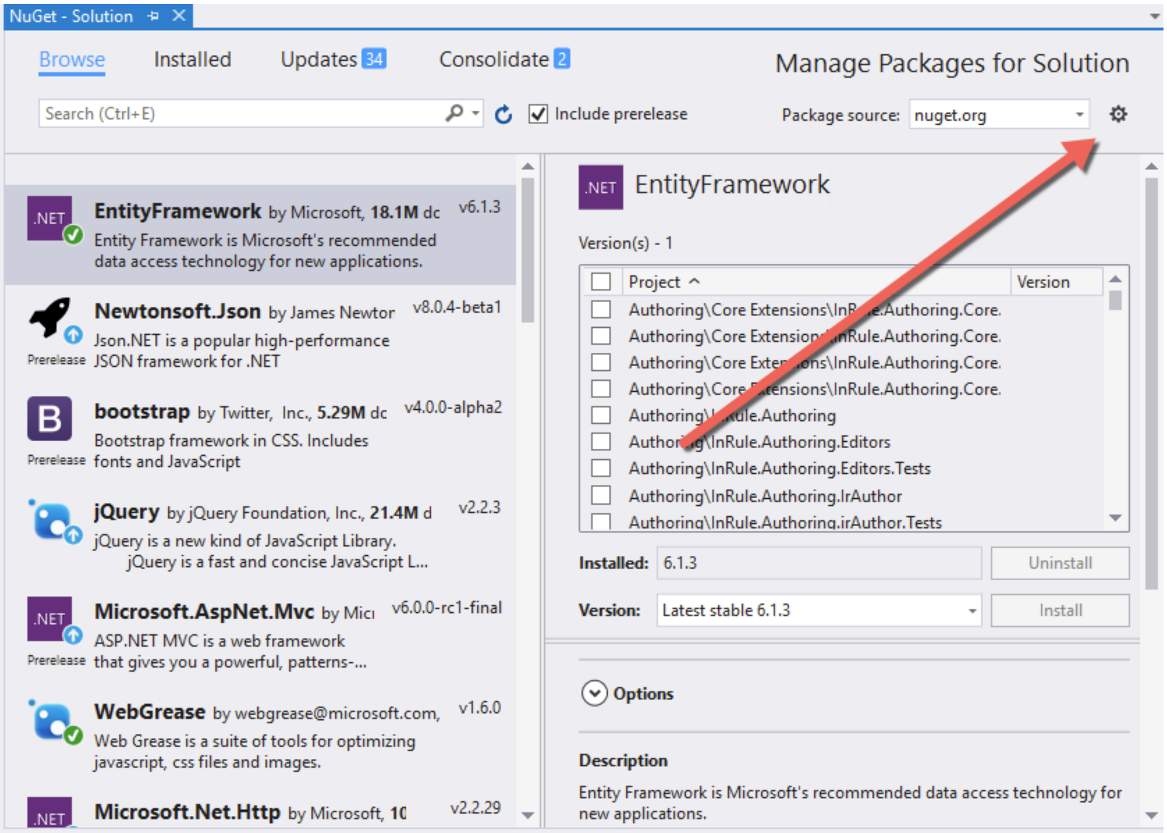
Versioning

The package versions will match the first three digits of the InRule product version number. Only the first three digits are used to conform to the Semantic Versioning rules enforced by NuGet.

Consuming the Packages

While you may choose to publish the packages to your own private NuGet server, InRule does not publish the packages to a public NuGet repository. If you do not wish to host your own NuGet server (or use a third party service to host a private server), you can use a file folder as a NuGet source.

In Visual Studio, you can do this by going to the NuGet Package Manager and clicking the gear icon next to the Package Source list, or going to the Visual Studio Options and selecting "Package Sources" under "NuGet Package Manager"


NuGet - Solution

Browse   Installed   Updates 34   Consolidate 2   Manage Packages for Solution

Search (Ctrl+E)   ☑ Include prerelease   Package source: nuget.org

EntityFramework by Microsoft, **18.1M** dc   v6.1.3
Entity Framework is Microsoft's recommended data access technology for new applications.

Newtonsoft.Json by James Newtor   v8.0.4-beta1
Prerelease   Json.NET is a popular high-performance JSON framework for .NET

bootstrap by Twitter, Inc., **5.29M** dc   v4.0.0-alpha2
Prerelease   Bootstrap framework in CSS. Includes fonts and JavaScript

jQuery by jQuery Foundation, Inc., **21.4M** d   v2.2.3
jQuery is a new kind of JavaScript Library. jQuery is a fast and concise JavaScript L...

Microsoft.AspNet.Mvc by Mici   v6.0.0-rc1-final
Prerelease   ASP.NET MVC is a web framework that gives you a powerful, patterns-...

WebGrease by webgrease@microsoft.com,   v1.6.0
Web Grease is a suite of tools for optimizing javascript, css files and images.

Microsoft.Net.Http by Microsoft, 10   v2.2.29

EntityFramework

Version(s) - 1

□ Project	Version
□ Authoring\Core Extensions\InRule.Authoring.Core.	
□ Authoring\Core Extensions\InRule.Authoring.Core.	
□ Authoring\Core Extensions\InRule.Authoring.Core.	
□ Authoring\Core Extensions\InRule.Authoring.Core.	
□ Authoring\InRule.Authoring	
□ Authoring\InRule.Authoring.Editors	
□ Authoring\InRule.Authoring.Editors.Tests	
□ Authoring\InRule.Authoring.IrAuthor	
□ Authoring\InRule.Authoring.irAuthor.Tests	

Installed: 6.1.3   Uninstall

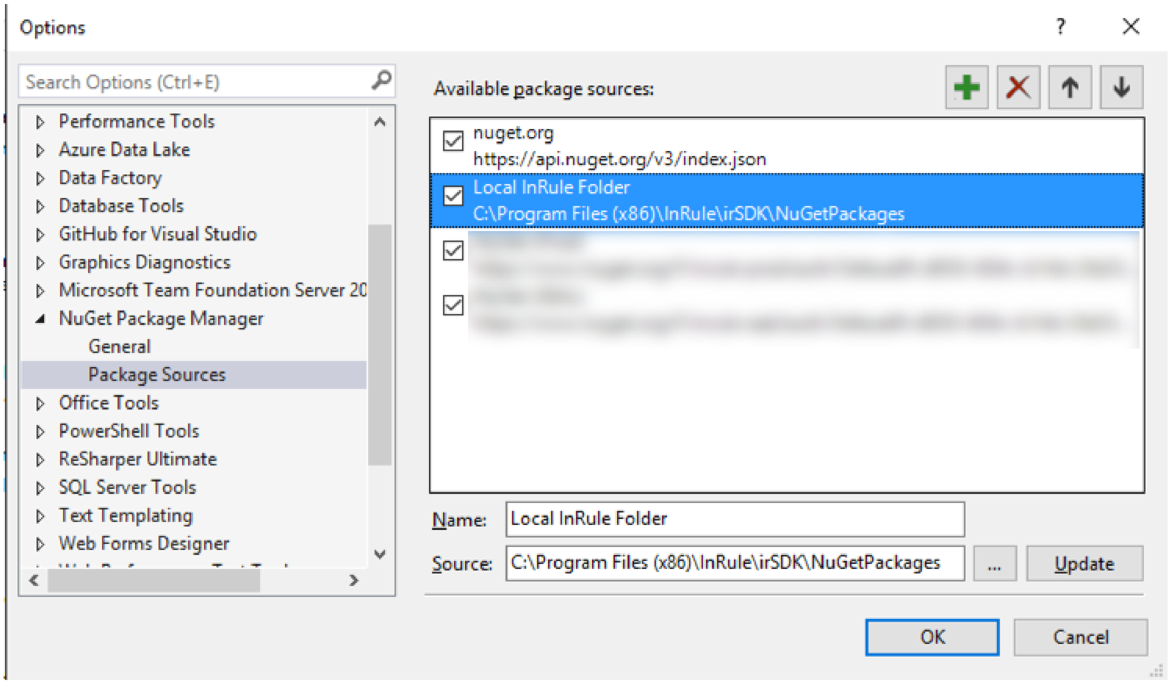
Version: Latest stable 6.1.3   Install

Options

Description

Entity Framework is Microsoft's recommended data access technology for new applications.

This will open the NuGet package sources options. Click the "+" icon to add a new source, then in the "Source" field, use the "..." button to browse to the folder holding the NuGet packages:


Options

Search Options (Ctrl+E)

- Performance Tools
- Azure Data Lake
- Data Factory
- Database Tools
- GitHub for Visual Studio
- Graphics Diagnostics
- Microsoft Team Foundation Server 20
- NuGet Package Manager
  - General
  - Package Sources
- Office Tools
- PowerShell Tools
- ReSharper Ultimate
- SQL Server Tools
- Text Templating
- Web Forms Designer

Available package sources:

☑ nuget.org
https://api.nuget.org/v3/index.json

☑ Local InRule Folder
C:\Program Files (x86)\InRule\irSDK\NuGetPackages

☑ [redacted]

☑ [redacted]

Name: Local InRule Folder

Source: C:\Program Files (x86)\InRule\irSDK\NuGetPackages   ...   Update

OK   Cancel

Click "Update" and "OK" to save the changes. Back in the NuGet Package Manager, in the "Package Sources" drop-down, select the new location (or "All") to include the packages in that folder in your NuGet repository.

4.5 Configuration Files

A variety of adjustable parameters or settings that are utilized by InRule are available as sections or keys that can be added to configuration files. These settings govern the behavior of specific functionality of InRule components.

The following topics describe InRule config file sections:

- InRule Runtime Config File Settings
- InRule Catalog Service Config File Settings
- InRule Authoring/Client Config File Settings
- InRule Rutime Service Config File Settings

The InRule configuration files can be found at the following locations:

- **Catalog Service**
  - IIS - <InRule installation directory>\irServer\RepositoryService\IisService\Web.config
  - Windows Service - <InRule installation directory>\irServer\RepositoryService\WindowsService\RepositoryWindowsService.exe.config

- **Rule Engine Service**
  - IIS - <InRule installation directory>\irServer\RuleEngineService\IisService\Web.config
  - Windows Service - <InRule installation directory>\irServer\RuleEngineService\WindowsService\RepositoryWindowsService.exe.config

- **irAuthor and irVerify**
  - <InRule installation directory>\irAuthor\irAuthor.exe.config

4.5.1 InRule Runtime Config File Settings

Configuration Settings for Common Production Deployment Scenarios

InRule includes two configuration file sections that affect memory use, rule execution behavior, timeouts, and licensing. The two configuration sections are inrule.runtime and inrule.repository. Most production deployments have both of these sections defined for processes that are hosting the rule engine. The settings for both sections are described below.

Alternatively many of these settings can be configured using AppSettings. If a value is set in both AppSettings and the configuration sections, the AppSettings value will take precedence.

In addition to the summary tables below, the following links provide more detail about specific config file settings in these sections:

- InRule Logging Config File Settings
- Rule Application Cache Settings
- Data Query Cache
- End Point Assemblies Folder
- Execute Query Timeout
- .NET Framework Runtime Config Settings With InRule

For settings that affect logging behavior for the runtime please see InRule Logging Config File Settings.

inrule.runtime Config Section/AppSettings

The runtime settings allow specification of the parameters for runtime rule processing.

Sub Section	Setting Name	AppSettings Key	Description
catalogRuleApplication	connectionTimeout	inrule:runtime:catalogRuleApplication:connectionTimeout	(TimeSpan) The timeout value for a successful Catalog Service connection. If the timeout expires without a successful connection to the Catalog, an exception will be thrown. Default is 00:01:00 (one minute).
	refreshInterval	inrule:runtime:catalogRuleApplication:refreshInterval	(TimeSpan) The time interval for refreshing a cached Rule Application Reference from the Catalog. Default is 00:00:30 (30 seconds).
	enableBackgroundCompilation	inrule:runtime:catalogRuleApplication:enableBackgroundCompilation	(boolean) If true, the polling operation to the Catalog to see if a newer rule application version is available and, if applicable, the download, compilation and caching of the new version will be performed on a background thread. Default is false.
ruleEngine	captureLockOwnerStackTrace	inrule:runtime:ruleEngine:captureLockOwnerStackTrace	(boolean) If true, the engine captures the stack trace of a thread that has become deadlocked during a Rule Application compile. Default is false.
	compiledApplicationCacheDepth	inrule:runtime:ruleEngine:compiledApplicationCacheDepth	(integer) Determines the number of unique compiled rule applications that are kept in memory. If this number is exceeded, the oldest compiled rule application is removed from the cache in favor on the newest compiled app. Default is 25.
	compileLockAcquisitio	inrule:runtime:ruleEngine:co	(TimeSpan) In case of a

	nTimeout	mpileLockAcquisitionTimeout	deadlock during a Rule Application compile, the amount of time before an exception will be thrown to break the deadlock. Default is 00:30:00 (30 minutes).
	enableBackgroundCompilation	inrule:runtime:ruleEngine:enableBackgroundCompilation	(boolean) If true, the compilation and caching of a new version, if applicable, will be performed on a background thread. Default is false.

Example:

```
    <appSettings>
      <add key="inrule:runtime:catalogRuleApplication:compiledApplicationCacheDepth"
  value="25">
      <add key="inrule:runtime:catalogRuleApplication:refreshInterval"
  value="00:00:30">
    </appSettings>
```

OR

```
    <configuration>
      <configSections>
        <section name="inrule.runtime"
    type="InRule.Runtime.Configuration.RuntimeConfigSectionHandler, InRule.Runtime" /
    >
        ...
      </configSections>
      ...
     <inrule.runtime>
         <ruleEngine compiledApplicationCacheDepth="25" />
         <catalogRuleApplication refreshInterval="00:00:30"
         cachePurgeInterval="00:10:00" />
     </inrule.runtime>
```

inrule.repository Config Section/AppSettings

The repository settings allow specification of the parameters that impact both authoring time and runtime behavior.

Sub Section	Setting Name	AppSettings Key	Description
assemblyEndPoint	endPointAssemblyPath	inrule:repository:endPoints:assemblyEndPoint:endPointAssemblyPath	(string) Denotes the relative path to the directory that contains assemblies referenced within various rule applications. This location is important because it allows the rule engine to load. Default is "EndPointAssemblies",

			which is relative from the executing assembly directory.
licensing	licenseFolder	inrule:repository:licensing:licenseFolder	(string) Determines the custom directory where InRule will look to verify the license file. The search order for the license file is as follows: 1) Current directory where application is running 2) Bin directory (which may be the same as #1) 3) AppData directory (which usually maps to C:\ProgramData\InRule\SharedLicenses) 4) Directory defined by this configuration setting **NOTE:** This is the path to the directory containing the license file, NOT the file itself. Default is null.

Example:

```
 <appSettings>
  <add key="inrule:repository:endPoints:assemblyEndPoint:endPointAssemblyPath"
value="EndPointAssemblies">
  <add key="inrule:repository:licensing:licenseFolder " value="..\LicenseFile">
 </appSettings>
```

OR

```
<configuration>
  <configSections>
    <section name="inrule.repository"
type="InRule.Repository.Configuration.RepositoryConfigSectionHandler,
InRule.Repository" />
    ...
  </configSections>
  ...
  <inrule.repository>
    <endPoints>
      <assemblyEndPoint
                endPointAssemblyPath="EndPointAssemblies" />
    </endPoints>
    <licensing licenseFolder="..\LicenseFile" />

  </inrule.repository>
```

4.5.1.1 InRule Logging Config File Settings

Detailed application event information from InRule can be logged to the Windows Application Event Log by configuring the following section in a client config file:

```
    <configuration>
      <configSections>
        <section name="inrule.logging"
type="InRule.Repository.Logging.Configuration.LoggingSectionHandler,
InRule.Repository" />
        ...
      </configSections>
      ...
      <inrule.logging>
        <group typeName="InRule.Repository.Logging.Loggers.LoggerGroup,
InRule.Repository" level="Warn">
          <logger (see below)/>
        </group>
      </inrule.logging>
```

Group level settings can be one of: *Debug, Info, Warn, Error, Fatal*

Logger level settings:

Group level settings for all loggers can be set with AppSettings*:*

```
     <appSettings>
       <add key="inrule:logging:level" value="Warn">
     </appSettings>
```

Adding this setting will override all level settings for loggers.

Logger type settings:

- ***EventLogLogger*** - records to the InRule event log. If the eventSource value is not specified, it uses the InRule source.

```
          <logger typeName="InRule.Repository.Logging.Loggers.EventLogLogger,
InRule.Repository" >
            <option name="eventSource" value="InRule" />
          </logger>
```

- ***FileLogger*** - records to a file. If "filename" option is not specified, it records to %TEMP%\InRuleFileLoggerLogs. FileLogger has the following parameters

```
          <logger typeName="InRule.Repository.Logging.Loggers.FileLogger,
InRule.Repository">
            <option name="filename" value="%TEMP%\YourApp.log"/>
          </logger>
```

- ***XmlLogger*** is also available as an alternative to FileLogger

- ***LibraryLogger*** - allows logging of application event information from InRule to 3rd party loggers including: Log4Net, Loupe, NLog and Serilog.

```
   <inrule.logging>
      <group typeName="InRule.Repository.Logging.Loggers.LoggerGroup,
InRule.Repository" level="Warn">
         <logger typeName="InRule.Repository.Logging.Loggers.LibraryLogger,
InRule.Repository" />
      </group>
   </inrule.logging>
```

Notes

- For production SDK app deployments, an EventLogLogger is automatically created and used (if no inrule.logging config entries present) if the InRule event log source(s) are registered, otherwise a FileLogger is created and used to the default logfile location above.
- InRule event logging should not be confused with RuleExecutionLog which returns rule execution information in a rule session.
- A setting of Debug should only be used for true debugging scenarios as it can slow overall performance with the amount of information returned

Diagnostic settings for WCF Logging

If a WCF issue is suspected with an InRule service, WCF tracing is available to further diagnose communication issues. WCF tracing is built on top of System.Diagnostics, which provides classes that allow you to interact with system processes, event logs,and performance counters. WCF tracing can be configured using the following config file settings.

Note: This should only be used for debugging as it can significantly reduce performance.

Example:

```
<configuration>
...
<system.diagnostics>
  <sources>
    <source name="System.ServiceModel"
        switchValue="Information, ActivityTracing"
        propagateActivity="true">
      <listeners>
        <add name="traceListener"
                type="System.Diagnostics.XmlWriterTraceListener"
                initializeData= "c:\log\Traces.svclog" />
      </listeners>
    </source>
  </sources>
</system.diagnostics>
...
</configuration>
```

irAuthor activity logging settings

An audit trail of user activities can be captured while authoring rules inside of irAuthor. This logging information can be used in order to provide information to InRule in the case of an error situation. The following settings can be used to configure how the logging will be managed. Log files are stored in the system defined temp directory on the author's machine.

Example:

```
    <configuration>
      <configSections>
        <section name="inrule.authoring"
type="InRule.Authoring.Configuration.AuthoringConfigSectionHandler,
InRule.Authoring.UI" />
        ...
      </configSections>
      ...
      <inrule.authoring>
        <tracing logFileCleanUpInterval="30.00:00:00" />
      </inrule.authoring>
      ...
    </configuration>
```

Info level logging SDK configuration

There are three types of Info level log messages that can be emitted from the RuleSession:

- CreateEntity
- CreateDecision
- ExecuteRules

These types can be configured at a granular level via the RuleSession.Settings.InfoLevelLogging property. This is a [Flags] enum type so multiple combinations may be applied as needed.
If Info level logging is enabled in the .config file, then all three will be applied by default. If a lower level logging is enabled (e.g. Warn, Error), then none of them will be applied by default.
The granular types may be added/removed regardless of the setting in the .config file.

Example showing how to remove CreateEntity logging and only include ExecuteRules logging if Info level logging is enabled in the .config:
(Note the ^= operator removes the proceeding enum value if it exists in the current property value)

```
using (var session = new RuleSession(ra))
{
      session.Settings.InfoLevelLoggingTypes ^= InfoLevelLoggingTypes.CreateEntity;

      var invoice = session.CreateEntity("Invoice");
      session.ApplyRules();
}
```

Example showing how to add ExecuteRules logging if Info level logging is disabled in the .config:
(Note the |= operator adds the proceeding enum value if it does not exist in the current property value)

```
using (var session = new RuleSession(ra))
{
      session.Settings.InfoLevelLoggingTypes |= InfoLevelLoggingTypes.ExecuteRules;

      var invoice = session.CreateEntity("Invoice");
      session.ApplyRules();
}
```

4.5.1.2 Rule Application Cache Settings

The first time a Rule Application is requested, it must be compiled so the rules engine can execute the rules. Once a Rule Application is compiled, it is stored in an AppDomain cache for subsequent executions of the application.

The cache, by default, may store 25 Rule Applications at a time. Each time a Rule Application is requested, the rule engine will first check to see if the Rule Application is in the cache. When a Rule Application is requested that is not in the cache, it will be retrieved, compiled and then cached. If the cache limit has been exceeded, one of the Rule Applications in the cache will be removed.

The cache depth can be modified by adding the following to the application configuration file:

```
 <appSettings>
   <add key="inrule:runtime:ruleEngine:compiledApplicationCacheDepth" value="25">
 </appSettings>
```

OR

```
<configuration>
  <configSections>
    <section name="inrule.runtime"
type="InRule.Runtime.Configuration.RuntimeConfigSectionHandler, InRule.Runtime" />
    ...
  </configSections>
  ...
  <inrule.runtime>
    <ruleEngine compiledApplicationCacheDepth="25" />
    ...
  </inrule.runtime>
</configuration>
```

FileSystemRuleApplicationReference, CatalogRuleApplicationReference and InMemoryRuleApplicationReference all leverage caching. The methods with which the cache is checked are as follows:

- ***InMemoryRuleApplicationReference***: A checksum is performed to see if the Rule Application has changed. This check is more expensive than both the file system check and lightweight Catalog check, which is why it is better to leverage FileSystemRuleApplicationReference or CatalogRuleApplicationReference unless you are modifying rules at runtime.

- ***FileSystemRuleApplicationReference***: A check is performed to see if the Rule Application file timestamp has been modified.

- ***CatalogRuleApplicationReference***: When a Rule Application is pulled from the Catalog, it is cached on the client, not the Catalog Service. If the compiled Rule Application is still in the client cache and within the refresh interval when a new RuleSession is created with this reference, the cached Rule Application is used. If the refresh interval has expired, InRule performs a lightweight polling operation to the Catalog to see if a newer version is available. If a new version is available, the latest revision is downloaded, compiled and cached. The following configuration option may be used to change the default refresh interval:

```
     <appSettings>
        <add key="inrule:runtime:catalogRuleApplication:refreshInterval"
value="00:00:30">
     </appSettings>
```

OR

```
         <configuration>
           <configSections>
           <section name="inrule.runtime"
type="InRule.Runtime.Configuration.RuntimeConfigSectionHandler, InRule.Runtime" />
           ...
           </configSections>
           ...
           <inrule.runtime>
             <catalogRuleApplication refreshInterval="00:00:30" />
             ...
           </inrule.runtime>
         </configuration>
```

Note: This setting may be overridden via the SDK on a per Rule Application basis via CatalogRuleApplicationReference.RefreshSettings.

By default, the check and recompilation of File System and Catalog Rule Applications will be performed on the main thread. The following configuration option may be used to change the compilation behavior to check and recompile on a background thread:

```
     <appSettings>
        <add key="inrule:runtime:ruleEngine:enableBackgroundCompilation"
value="true">
     </appSettings>
```

OR

```
         <configuration>
           <configSections>
           <section name="inrule.runtime"
type="InRule.Runtime.Configuration.RuntimeConfigSectionHandler, InRule.Runtime" />
           ...
           </configSections>
           ...
           <inrule.runtime>
             <ruleEngine enableBackgroundCompilation="true" />
             ...
           </inrule.runtime>
         </configuration>
```

Note: This setting may be overridden via the SDK on a per Rule Application basis via FileSystemRuleApplicationReference.EnableBackgroundCompilation or CatalogRuleApplicationReference.EnableBackgroundCompilation. The setting on InMemoryRuleApplicationReference is ignored.

There is some overhead involved when a Rule Application is retrieved, compiled, and added to the cache which will not be present on subsequent RuleSession creations.

4.5.1.3 Data Query Cache

The InRule Runtime can cache the results of data queries to reduce latency of querying external data sources that can change infrequently.

The following InRule Queries may be cached:

- Lookup() function

- TableLookup() function
- QueryToList() function
- IsInValueList() function
- ValueListLookup() function
- SQL query functions
- Execute SQL Query action
- Execute REST Service action

Query cache results are cached for any RuleSession in the current AppDomain. Queries that are identical in their parameters across different RuleSessions and source Rule Applications will use the same cached results.

The following query parameters are taken into account when comparing identity:

- Data source, e.g. database connection string, REST service URL
- Query text
- Parameter values
- Cache timeout

Data Elements

How the query results are cached depends on the type of Data Element used in the query, and on any cache settings that may have been configured. The following illustrates the default cache timeout values for different Data Elements; most of them are configurable.

Lookup and TableLookup functions

Target	Timeout	Configurable
Inline Table	300 seconds	no
Linked Table	300 seconds	yes

IsInValueList and ValueListLookup functions

Target	Timeout	Configurable
Query ValueList: Inline Table	300 seconds	no
Query ValueList: Linked Table	300 seconds	yes
Query ValueList: SQL Query	0 seconds	yes

SQL Query function, Execute SQL Query action, and QueryToList() function

Target	Timeout	Configurable
SQL Query	0 seconds	yes

Execute REST Service action

Target	Timeout	Configurable
REST Operation	0 seconds	yes

Any Data Element configured with a zero second cache timeout will not enter the cache, which will result in the target data source being queried each time.

Inline Tables queried by syntax functions will remain in the cache for 300 seconds (five minutes). the underlying data cannot change (unless an override is applied), so they are kept for 300 seconds. The query results are not kept indefinitely in order to allow memory to eventually be reclaimed if the host process does not perform any inline Table queries for an extended period of time.

The cache timeout on an SQL Query will override the setting of the target Inline Table or Linked Table.

Cache Configuration

The underlying cache mechanism uses the System.Runtime.Caching.MemoryCache implementation in the .NET Framework, which may be familiar to users of ASP.NET.

While cached query results may be configured to expire from the cache after a period of time, the cache itself may evict query results when memory pressure is detected, to attempt to avoid an OutOfMemoryException. The default settings cause the cache to poll memory usage every two minutes, and start evicting query results when the cache uses approximately 60% of physical RAM.

If memory pressure is detected, the cache will start evicting query results using a Least-Recently-Used (LRU) algorithm.

To change the default cache memory limit settings, add the following configuration element to the application's .config file:

```
 <appSettings>
    <add key="inrule:runtime:dataQueryCache:pollingInterval" value="00:00:10">
    <add key="inrule:runtime:dataQueryCache:cacheMemoryLimitMegabytes" value="500">
    <add key="inrule:runtime:dataQueryCache:physicalMemoryLimitPercentage"
value="20">
 </appSettings>
```

OR

```
    <configuration>
      <configSections>
        <section name="inrule.runtime"
            type="InRule.Runtime.Configuration.RuntimeConfigSectionHandler,
InRule.Runtime" />
        ...
      </configSections>
      ...
      <inrule.runtime>
        <dataQueryCache
            pollingInterval="00:00:10"
            cacheMemoryLimitMegabytes="500"
            physicalMemoryLimitPercentage="20" />
      </inrule.runtime>
    </configuration>
```

The above example changes the polling interval to 10 seconds, sets a cache memory limit of 500MB, and sets a physical memory limit of 20%. If only one limit is used, either set the other limit to zero or omit it altogether.

In practice, memory pressure detection does not trigger until a Generation 2 Garbage Collection occurs.

If the Cache Memory Limit is used, the .NET Framework does not enforce this value as a hard limit; the memory used may significantly exceed the configured valu8e until a Generation 2 Garbage Collection occurs, at which point the cache will be trimmed to the configured value.

4.5.1.4 EndPoint Assemblies Folder

.NET assemblies that are referenced from rule applications must be accessible to the rule engine for execution. Referenced assemblies could include static function libraries, bound business objects, and referenced type libraries. InRule will check several locations, in order, to locate any referenced assemblies.

For in-process connections, the default folder locations for InRule to check for assemblies during execution include the following:

- irVerify: InRule\irAuthor\EndPointAssemblies
- Custom .NET application: the application's bin directory

For out-of-process connections, the default folder locations for InRule to check for assemblies during execution include the following:

- IIS: InRule\irServer\RuleEngineService\IisService\bin\EndPointAssemblies
- Windows Service: InRule\irServer\RuleEngineService\WindowsService\EndPointAssemblies

Other locations where assemblies may reside include the following:

- Global Assembly Cache (GAC)
- A user defined directory which is designated by adding the following setting to the application config file:

```
 <appSettings>
    <add key="inrule:repository:endPoints:assemblyEndPoint:endPointAssemblyPath"
value="EndPointAssemblies">
 </appSettings>
```

OR

```
    <configuration>
      <configSections>
        <section name="inrule.repository"
type="InRule.Repository.Configuration.RepositoryConfigSectionHandler,
InRule.Repository" />
        ...
      </configSections>
      ...
      <inrule.repository>
        <endPoints>
          <assemblyEndPoint endPointAssemblyPath="EndPointAssemblies" />
        </endPoints>
      ...
      </inrule.repository>
```

InRule checks folder locations for assemblies in the following order:

1. Global Assembly Cache (GAC)
2. Location specified in config file
3. EndPointAssemblies folder
4. The calling application's binaries directory (.NET only)

4.5.1.5 Execute Query Timeout

Any database query executed using the Execute SQL Query Action has a default timeout of 30 seconds.This timeout can be modified by adding the following to the client configuration file:

```
 <appSettings>
    <add key="inrule:runtime:dbCommand:dbCommandTimeout" value="50">
    </appSettings>
```

OR

```
    <configuration>
      <configSections>
        <section name="inrule.runtime"
type="InRule.Runtime.Configuration.RuntimeConfigSectionHandler, InRule.Runtime" />
        ...
      </configSections>
      ...
      <inrule.runtime>
        <dbCommand commandTimeout="50" />
        ...
      </inrule.runtime>
```

This commandTimeout maps to the standard .Net IDbCommand.CommandTimeout property and its value is specified in seconds. In the above example, a wait time of 50 seconds is set, before terminating to execute a command and generate an error.

4.5.1.6 .NET Framework Runtime Config Settings With InRule

The framework runtime settings give you the ability to specify the parameters of the Framework runtime components, such as assembly redirection. Refer to .Net Framework documentation for further information about framework runtime settings, which can be found here.

Note that for best performance in multi-core, free-threaded loading of the rule engine, the gcServer element should have enabled set to "true".

Example

```
<configuration>
  ...
  <runtime>
   <gcServer enabled="true" />
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity
                 name="Oracle.DataAccess"
                 publicKeyToken="89b483f429c47342" />
        <bindingRedirect
                 oldVersion="2.0.0.0-99.0.0.0"
                 newVersion="2.102.2.20"/>
        <!-- If using an ODP.NET endpoint, set this to version of ODP.NET you have
installed. -->
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
...
</configuration>
```

4.5.2 InRule Catalog Service Config File Settings

Catalog Service Settings

The Catalog settings allow specification of the database and security settings for the Catalog service.

Many of these settings can be configured using AppSettings. If a value is set in both AppSettings and the configuration sections, the AppSettings value will take precedence.

The InRule catalog service configuration files can be found at the following locations:

- IIS - <InRule installation directory>\irServer\RepositoryService\IisService\Web.config
- Windows Service - <InRule installation directory>\irServer\RepositoryService \WindowsService\RepositoryWindowsService.exe.config

For settings that affect logging behavior for the catalog service please service see InRule Logging Config File Settings.

inrule.repository.service Config Section/AppSettings

Setting Name	AppSettings Key	Description
uri	N/A	(string) Denotes the URL address where the Catalog service will be published. **Note:** this value will be set by the InRule Configuration wizard during install time.
connectionString	inrule:repository:service:connectionString	(string) The connection string to the catalog database. Note that for SQL Server and Oracle, the specific ADO.NET data providers are used, so they do not need to be included in the connection string. **Note:** this value will be set by the InRule Configuration wizard during install time.
authentication	N/A	(string) The "type" attribute determines which store the service will access when authenticating users. Possible values are: • Database -- the user store contained within the catalog database schema • LDAP -- another user store that supports LDAP authentication, such as Active Directory • Custom **Note:** this value will be set by the InRule Configuration wizard during install time.
catalogType	N/A	(string) The type of database service that is hosting the catalog database. Possible values are: • SqlServer -- catalog hosted in Microsoft SQL Server • MsOracle -- catalog hosted in Oracle 9 or later with Microsoft Oracle ADO.NET driver for catalog calls • OdpOracle -- catalog hosted in Oracle 9 or later with ODP Oracle ADO.NET driver for catalog calls • Embedded -- reserved for future use **Note:** this value will be set by the InRule Configuration wizard during install time.

Example:

```
<configSections>
  <section name="inrule.repository.service"
type
=
"InRule.Repository.Service.Configuration.ConfigSectionHandler,InRule.Repository.S
ervice, PublicKeyToken=1feb8dd25b1ceb6b" />
  ...
</configSections>
...
<inrule.repository.service>
        <uri>http://someuriaddress:8082/InRuleCatalogService</uri>
        <connectionString>Server=DBServerName
\InstanceName;Database=InRuleCatalog;Trusted_Connection=yes</connectionString>
        <authentication type="Database" />
        <catalogType>SqlServer</catalogType>
</inrule.repository.service>
```

4.5.2.1 Additional Catalog Service Config Settings

The InRule catalog service has several relevant options that are adjustable in the configuration file for the service.

The InRule catalog service configuration files can be found at the following locations:

- IIS - <InRule installation directory>\irServer\RepositoryService\IisService\Web.config
- Windows Service - <InRule installation directory>\irServer\RepositoryService\WindowsService\RepositoryWindowsService.exe.config

Catalog Service configuration settings

- Database Path
- Maximum Number of Rule Elements
- Lock Acquisition Timeout In Seconds
- Command Batch Size
- Enable Repository Service Aggregate Stats
- Enable XACT_ABORT Support

Database Path

Database connection information for the catalog service is controlled by the settings contained in the element entitled *<inrule.repository.service>*. The most common adjustment of this configuration setting is to alter the database name to be utilized if irCatalog is installed in multiple locations. For example:

```
   <configuration>
      <configSections>
        <section name="inrule.repository.service"
type
=
"InRule.Repository.Service.Configuration.ConfigSectionHandler,InRule.Repository.Servic
e" />
        ...
      </configSections>
      ...
      <inrule.repository.service>
        <uri>http://localhost/InRuleCatalogService</uri>
        <connectionString>Server=.\SQLEXPRESS;Database=InRuleCatalog;Integrated
Security=SSPI</connectionString>
        <authentication type="Database" />
        <catalogType>SqlServer</catalogType>
      </inrule.repository.service>
      ...
   </configuration>
```

Maximum Number of Rule Elements

The *MaxItemsInObjectGraph* parameter specifies the maximum number of rule elements that can be saved to or retrieved from the catalog. The *MaxItemsInObjectGraph* parameter can be specified as follows:

```
    <appSettings>
       <add key="inrule:repository:service:maxItemsInObjectGraph" value="2147483647">
    </appSettings>
```

OR

```
     <configuration>
        <configSections>
          <section name="inrule.repository.service"
type
=
"InRule.Repository.Service.Configuration.ConfigSectionHandler,InRule.Repository.Servic
e" />
          ...
        </configSections>
        ...
        <inrule.repository.service>
          <MaxItemsInObjectGraph>2147483647</MaxItemsInObjectGraph>
        </inrule.repository.service>
        ...
     </configuration>
```

Lock Acquisition Timeout In Seconds

The *LockAcquisitionTimeoutInSeconds* parameter specifies the maximum period of time to wait for a lock acquisition to complete. The LockAcquisitionTimeoutInSeconds parameter can be specified as follows:

```
    <appSettings>
       <add key="inrule:repository:service:lockAcquisitionTimeoutInSeconds"
   value="100">
    </appSettings>
```

OR

```
     <configuration>
        <configSections>
          <section name="inrule.repository.service"
type
=
"InRule.Repository.Service.Configuration.ConfigSectionHandler,InRule.Repository.Servic
e" />
          ...
        </configSections>
        ...
        <inrule.repository.service>
          <lockAcquisitionTimeoutInSeconds>100</lockAcquisitionTimeoutInSeconds>
        </inrule.repository.service>
        ...
     </configuration>
```

Command Batch Size

The *CommandBatchSize* parameter specifies the maximum number of sql statements per batch, when it is submitted to the database server. The following is an example of the setting configuration with this parameter specified:

```
    <appSettings>
       <add key="inrule:repository:service:commandBatchSize" value="40">
    </appSettings>
```

OR

```
     <configuration>
        <configSections>
          <section name="inrule.repository.service"
type
=
"InRule.Repository.Service.Configuration.ConfigSectionHandler,InRule.Repository.Servic
e" />
          ...
        </configSections>
        ...
        <inrule.repository.service>
          <commandBatchSize>40</commandBatchSize>
        </inrule.repository.service>
        ...
     </configuration>
```

Enable Repository Service Aggregate Statistics

The *enableRepositoryServiceAggrStats* parameter specifies if aggregate statistics shall be included in the SOAP header for all Catalog responses. The following is an example of the setting configuration with this parameter specified:

```
    <appSettings>
       <add key="inrule:repository:service:enableRepositoryServiceAggrStats"
   value="true">
    </appSettings>
```

OR

```
     <configuration>
        <configSections>
          <section name="inrule.repository.service"
type
=
"InRule.Repository.Service.Configuration.ConfigSectionHandler,InRule.Repository.Servic
e" />
          ...
        </configSections>
        ...
        <inrule.repository.service>
          <enableRepositoryServiceAggrStats>true</enableRepositoryServiceAggrStats>
        </inrule.repository.service>
        ...
      </configuration>
```

Enable XACT_ABORT Support

The enableXActAbortSupport allows Catalog Upgrader to run when XACT_ABORT is enabled in SQL Server. The following is an example of the setting configuration with this parameter specified:

```
 <appSettings>
   <add key="inrule:repository:service:enableXActAbortSupport" value="false">
 </appSettings>
```

OR

```
      <configuration>
        <configSections>
          <section name="inrule.repository.service"
type
=
"InRule.Repository.Service.Configuration.ConfigSectionHandler,InRule.Repository.Servic
e" />
          ...
        </configSections>
        ...
        <inrule.repository.service>
          <enableXActAbortSupport>true</enableXActAbortSupport>
        </inrule.repository.service>
        ...
      </configuration>
```

Default values

The following table specifies the default values for each setting:

Parameter name	**Default value**
MaxItemsInObjectGraph	2147483647
LockAcquisitionTimeoutInSeconds	30
CommandBatchSize	100
enableRepositoryServiceAggrStats	false
enableXActAbortSupport	false

4.5.3 InRule Authoring/Client Config File Settings

Authoring settings

The authoring settings allow to specify the parameters of the authoring experience, rule engine connectivity, and catalog connectivity.

The irAuthor configuration file can be found at <InRule installation directory>\irAuthor \irAuthor.exe.config.

For settings that affect logging behavior for authoring and the tester runtime please see the following link -- InRule Logging Config File Settings

inrule.authoring Config Section

Sub Section	Setting Name	Description
catalogClient	catalogServiceUri	(string) The URL address to the InRule Catalog service. Default is an empty string.
	defaultLoginName	(string) The default username that is used to connect to the InRule Catalog. Default is null.
ruleEngineClient	returnDetailStatisticsInfo	(boolean) Determines if irVerify will run rules with full detailed statistics reporting. Default is true.
	ruleEngineServiceUri	(string) The URL address to access the InRule Rule Engine Service. Default is null.
tracing	logFileCleanUpInterval	(TimeSpan) The maximum age of a log file. Expired files are deleted during irAuthor start-up. Default is "30.00:00:00" (30 days).

Example:

```
       <configSections>
              <section name="inrule.authoring"
type="InRule.Authoring.Configuration.AuthoringConfigSectionHandler,
InRule.Authoring.UI" />
              ...
       </configSections>
       ...
       <inrule.authoring>
              <catalogClient
                  catalogServiceUri="http://localhost:8082/InRuleRepositoryService"
                  defaultLoginName="username" />
              <tracing
                  logFileCleanUpInterval="30.00:00:00" />
       </inrule.authoring>
```

4.5.4 InRule Runtime Service Config File Settings

Runtime Service Settings

Specifies the root element for the Rule Execution Service configuration section and contains configuration elements that control how the Rule Execution Service behaves.

inrule.runtime.service Config Section/AppSettings

Sub Section	Setting Name	AppSettings Key	Description
runtime	catalogServiceUri	inrule:runtime:service:catalog:catalogServiceUri	(string) URI to the irCatalog Service. Default is empty string.
	userName	inrule:runtime:service:catalog:userName	(string) User name to be used when authenticating against irCatalog, if credentials were not passed to irServer Rule Execution Service. ***Only supported by the REST Endpoint.*** Default is empty string.
	password	inrule:runtime:service:catalog:password	(string) Password to be used when authenticating against irCatalog, if credentials were not passed to irServer Rule Execution Service. ***Only supported by the REST Endpoint.*** Default is empty string.

1. Locate the config file in the root of your irServer Rule Execution Service installation.
2. Navigate to the appSettings section
3. Add the key value pairs for catalogServiceUri, userName and password.

Example:

```
 <appSettings>
    <add key="inrule:runtime:service:catalog:catalogServiceUri" value="http://
localhost/InRuleCatalogService/Service.svc"/>
    <add key="inrule:runtime:service:catalog:userName"  value="userName"/>
    <add key="inrule:runtime:service:catalog:password" value="password"/>
 </appSettings>
```

OR

1. Locate the config file in the root of your irServer Rule Execution Service installation.
2. Navigate to the **inrule.runtime.service** configuration element.
3. Add the catalogServiceUri, userName and password attributes to the **runtime** element.

Example:

```
<configuration>
  <configSections>
    <section name="inrule.runtime.service"
type="InRule.Runtime.Service.Configuration.RuntimeServiceConfigSectionHandler, InRule.Runtime.S
    ...
  </configSections>
  ...
  <inrule.runtime.service>
    <catalog catalogServiceUri="http://localhost/InRuleCatalogService/Service.svc"
    userName="userName"
    password="password" />
  </inrule.runtime.service>
  ...
</configuration>
```

The irServer Rule Execution Service supports storing irCatalog credentials in the config file for use by irServer's REST endpoint. These credentials are used only when no credentials (Username / Password or Single Sign-On) are specified in the request. **Note: This is supported only for the irServer Rule Execution Service REST Endpoint.**

Endpoint and Data Element Overrides

Endpoint and Data Element overrides may also be specified via AppSettings. However, any overrides passed in the request will still take precedence over the AppSettings overrides.

The following Endpoints and Data Elements may be overridden by AppSettings:

AppSettings Key	Description
inrule:runtime:overrides:<endpoint-name>:DatabaseConnection:ConnectionString	(string) Database connection string.
inrule:runtime:overrides:<endpoint-name>:SendMailServer:ServerAddress	(string) Mail server host name.
inrule:runtime:overrides:<endpoint-name>:WebService:WsdlUri	(string) Web service WSDL URI.
inrule:runtime:overrides:<endpoint-name>:WebService:ServiceUriOverride	(string) Web service SOAP end point URI.
inrule:runtime:overrides:<endpoint-name>:WebService:WebServiceMaxReceivedMessageSize	(integer) Web service client max received message size in bytes. (max 2147483647)
inrule:runtime:overrides:<endpoint-name>:XmlDocumentPath:XmlPath	(string) XML document file path on Runtime Service.
inrule:runtime:overrides:<endpoint-name>:XmlSchema:XsdPath	(string) XML schema path or URL.
inrule:runtime:overrides:<endpoint-name>:XmlSchema:EnableXsdValidation	(boolean) Whether to validate XML Entity state against XSD. (true or false)
inrule:runtime:overrides:<endpoint-name>:RestService:RestServiceRootUrl	(string) REST service root URL.
inrule:runtime:overrides:<endpoint-name>:RestService:AuthenticationType	(string) REST service authentication type. (None, Basic, NTLM, Kerberos, Custom)
inrule:runtime:overrides:<endpoint-name>:RestService:RestServiceUserName	(string) REST service authentication username.
inrule:runtime:overrides:<endpoint-name>:RestService:RestServicePassword	(string) REST service authentication password.
inrule:runtime:overrides:<endpoint-name>:RestService:RestServiceDomain	(string) REST service authentication domain.
inrule:runtime:overrides:<endpoint-name>:RestService:RestServiceX509CertificatePath	(string) REST service X509 client certificate path on Runtime Service.
inrule:runtime:overrides:<endpoint-name>:RestService:RestServiceX509CertificatePassword	(string) REST service X509 client certificate password.
inrule:runtime:overrides:<endpoint-name>:RestService:RestServiceAllowUntrustedCertificates	(boolean) Whether to allow REST service certificates not signed by trusted CA. (true or false)
inrule:runtime:overrides:<dataElement-name>:SqlQuery:Query	(string) SQL query text.
inrule:runtime:overrides:<dataElement-	(string) XML serialized string of TableSettings

name>:InlineTable:TableSettings	object. (see 1.)
inrule:runtime:overrides:<dataElement-name>:InlineValueList:ValueListItems	(string) XML serialized string of ValueListItem collection (see 2.)
inrule:runtime:overrides:<dataElement-name>:InlineXmlDocument:InlineXml	(string) XML document.

1. The following C# code can be used with irSDK to create the XML serialized TableSettings from a DataTable:

```
TableSettings tableSettings = new TableSettings();
tableSettings.InlineDataTable.Columns.Add(new DataColumn("Column1", typeof(string)));
tableSettings.InlineDataTable.Columns.Add(new DataColumn("Column2", typeof(int)));
tableSettings.InlineDataTable.Rows.Add("One", 1);
tableSettings.InlineDataTable.Rows.Add("Two", 2);

StringBuilder sb = new StringBuilder();
using (XmlWriter writer = XmlWriter.Create(sb))
{
       XmlSerializer xs = new XmlSerializer(typeof(TableSettings));
       xs.Serialize(writer, tableSettings);
}

string serializedTableSettings = sb.ToString();
```

2. There is no irSDK data structure that can serialize the ValueListItems. The XML should be structured as follows:

```
<ValueListItems>
       <ValueListItem>
              <DisplayText>value one</DisplayText>
              <Value>value1</Value>
       </ValueListItem>
       <ValueListItem>
              <DisplayText>value two</DisplayText>
              <Value>value2</Value>
       </ValueListItem>
</ValueListItems>
```

Example:

```
<appSettings>
       <add
key="inrule:runtime:overrides:DatabaseConnection1:DatabaseConnection:ConnectionString
" value="Data Source = MyDbHost;Initial Catalog = Customer;Integrated Security =
SSPI;" />
       <add
key="inrule:runtime:overrides:SendMailServer1:SendMailServer:ServerAddress"
value="smtp.mycorp.com" />
       <add key="inrule:runtime:overrides:XmlSchema1:XmlSchema:EnableXsdValidation"
value="true" />
</appSettings>
```

Configuration Builders

AppSettings may be overridden by mechanisms other than the XML in the Rule Execution Service's

.config file.

For example, system Environment variables matching the AppSettings key names may be used to take precedence over the AppSettings defined in the XML .config file.
The Configuration Builders must be configured in the Rule Execution Service's .config file:

```
  <configSections>
    <section name="inrule.logging"
type="InRule.Repository.Logging.Configuration.LoggingSectionHandler,
InRule.Repository" />
    <section name="inrule.repository"
type="InRule.Repository.Configuration.RepositoryConfigSectionHandler,
InRule.Repository" />
    <section name="inrule.runtime"
type="InRule.Runtime.Configuration.RuntimeConfigSectionHandler, InRule.Runtime" />
    <section name="inrule.runtime.service"
type="InRule.Runtime.Service.Configuration.RuntimeServiceConfigSectionHandler,
InRule.Runtime.Service" />
    <section name="configBuilders"
type="System.Configuration.ConfigurationBuildersSection, System.Configuration,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
              restartOnExternalChanges="false"
              requirePermission="false" />
  </configSections>

  <configBuilders>
    <builders>
      <add name="Environment"
              mode="Greedy"
              name="inrule:runtime:overrides"

type="Microsoft.Configuration.ConfigurationBuilders.EnvironmentConfigBuilder,
Microsoft.Configuration.ConfigurationBuilders.Environment, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
    </builders>
  </configBuilders>

  <appSettings configBuilders="Environment">
  </appSettings>
```

4.6 Performance Logging and Monitoring

InRule provides the ability to analyze performance and other execution statistics with the following tools:

- Event Log Details
- Performance Statistics Report
- Rule Tracing

4.6.1 Event Log Details

Overview

When InRule is installed (or modified using the License Activation Utility) event logging can be set up to use either the Windows Application Event Log or an InRule Event Log source (the default is to use the InRule Event Log). InRule then uses either Event Log to record detailed application event information. This logging is performed any time irSDK is utilized, including the InRule products such as irAuthor, the Catalog Service and the Rule Engine Service. The default logging is set up to log Errors and Warnings, however, Info level logging can be configured for additional detail. For more information on how to fully configure Logging, see InRule Logging.

InRule Event Log Types

- Runtime Events
- Repository Events

4.6.1.1 Runtime Event Log Details

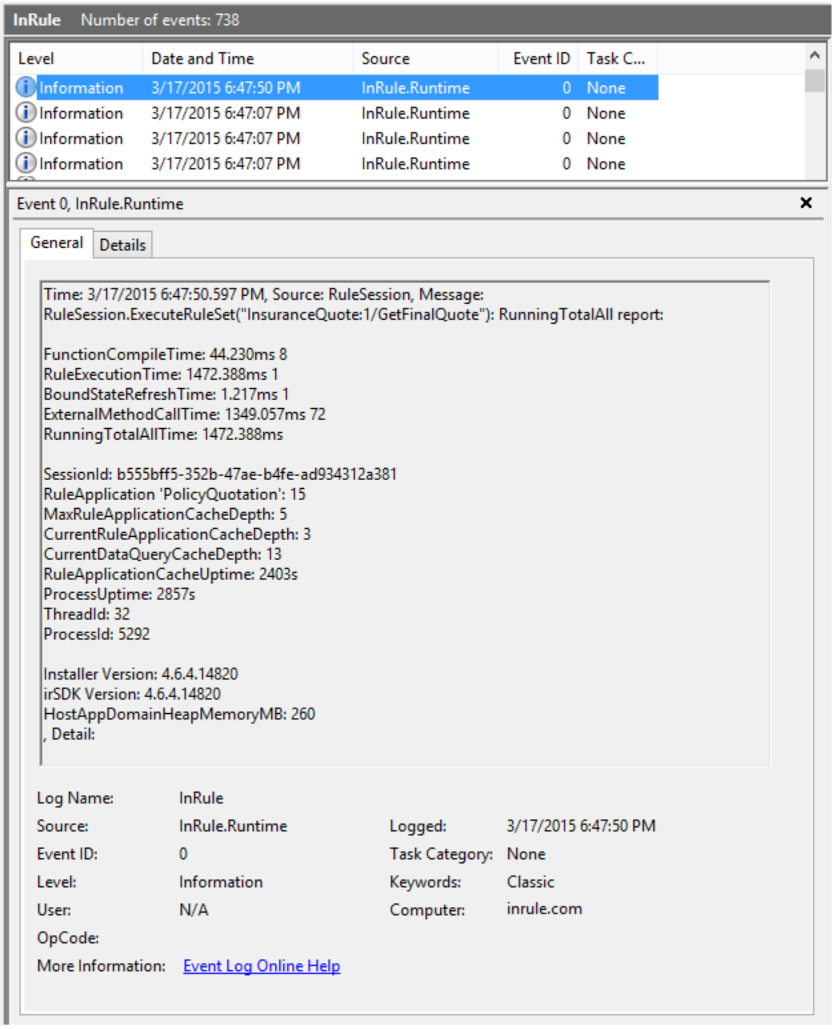
InRule.Runtime Event Log Details

The following 3 logging levels are available for InRule.Runtime:

- Error
- Warn
- Info

The default logging is set up to log Errors and Warnings only.

Below is an example of an InRule.Runtime "Info" level event log entry in the Windows Event Viewer.



InRule Number of events: 738

Level	Date and Time	Source	Event ID	Task C...
Information	3/17/2015 6:47:50 PM	InRule.Runtime	0	None
Information	3/17/2015 6:47:07 PM	InRule.Runtime	0	None
Information	3/17/2015 6:47:07 PM	InRule.Runtime	0	None
Information	3/17/2015 6:47:07 PM	InRule.Runtime	0	None

Event 0, InRule.Runtime

```
Time: 3/17/2015 6:47:50.597 PM, Source: RuleSession, Message:
RuleSession.ExecuteRuleSet("InsuranceQuote:1/GetFinalQuote"): RunningTotalAll report:

FunctionCompileTime: 44.230ms 8
RuleExecutionTime: 1472.388ms 1
BoundStateRefreshTime: 1.217ms 1
ExternalMethodCallTime: 1349.057ms 72
RunningTotalAllTime: 1472.388ms

SessionId: b555bff5-352b-47ae-b4fe-ad934312a381
RuleApplication 'PolicyQuotation': 15
MaxRuleApplicationCacheDepth: 5
CurrentRuleApplicationCacheDepth: 3
CurrentDataQueryCacheDepth: 13
RuleApplicationCacheUptime: 2403s
ProcessUptime: 2857s
ThreadId: 32
ProcessId: 5292

Installer Version: 4.6.4.14820
irSDK Version: 4.6.4.14820
HostAppDomainHeapMemoryMB: 260
, Detail:
```

Log Name:	InRule		
Source:	InRule.Runtime	Logged:	3/17/2015 6:47:50 PM
Event ID:	0	Task Category:	None
Level:	Information	Keywords:	Classic
User:	N/A	Computer:	inrule.com
OpCode:			
More Information:	Event Log Online Help		

4.6.1.1.1 Runtime Error Level Logging

InRule.Runtime Error Level Logging

Type	Name	Description
Runtime error	Message	Error description
	Installer Version	Version of the installer used to install InRule
	irSDK Version	Version of the InRule SDK
	HostAppDomainHeapMemoryMB	Current heap memory usage for the application domain associated with the running instance of InRule

		Note: this value is intermittently logged
	Processor Count	Number of processors/ cores
	CLR Version	Runtime version
	CLR Mode	32 or 64 bits
	Thread Culture	
	ThreadId	Managed identification numbers for the thread that wrote the log entry
	Error Information	Greater detail about the warning or exception

4.6.1.1.2 Runtime Warn Level Logging

InRule.Runtime Warn Level Logging

Type	Name	Description
Runtime warning	Message	Warning description
	Installer Version	Version of the installer used to install InRule
	irSDK Version	Version of the InRule SDK
	HostAppDomainHeapMemoryMB	Current heap memory usage for the application domain associated with the running instance of InRule **Note:** this value is intermittently logged
	Processor Count	Number of processors/ cores
	CLR Version	Runtime version
	CLR Mode	32 or 64 bits
	Thread Culture	
	ThreadId	Managed identification numbers for the thread that wrote the log entry
	Error Information	Greater detail about the warning or exception

4.6.1.1.3 Runtime Info Level Logging

InRule.Runtime Info Level Logging

The following event types are included in Info level logging:

- Compile
- Create Entity
- Apply Rules
- Execute Rule Set

4.6.1.1.3.1 Compile Event

InRule.Runtime Info Level Logging - Compile Event

Note: Some of the log detail items are optional and will not be present in every log entry.

Type	Name	Description
RuleApplication Compile	Time	The date and time of the event
	Source	The source of the event within InRule.Runtime (e.g. "CompiledRuleApplication")
	Message	The type of the event (e.g. "RuleApplication Compile")
	Is Background Compilation	Boolean value indicating if the compilation occurred on the main thread (false) or on the background thread (true)
	MetadataCompileTime	Elapsed time to validate rule application definition and generate fully resolved runtime meta model
	CatalogTotalClientCallTime	Elapsed time to complete both the Metadata compilation and the rule application download (MetadataCompileTime +

		CatalogRuleApplication DownloadTime)
	CatalogRuleApplicationVersionCheckTime	Elapsed time to poll the Catalog to check if there is a newer version of the rule application
	CatalogRuleApplicationDownloadTime	Elapsed time to download the new version of the rule application from the Catalog
	RunningTotalAllTime	Elapsed time to complete the Metadata compilation (MetadataCompileTime)
	SessionId	GUID representing a unique RuleSession
	RuleApplication	Name of the rule application, and associated revision number. A revision number of "-1" indicates a file based rule application where revision number is not applicable.
	MaxRuleApplicationCacheDepth	Maximum rule apps that will remain cached before rule apps are removed
	CurrentRuleApplicationCacheDepth	Current number of rule apps in the cache
	CurrentDataQueryCacheDepth	Current number of cached queries in the AppDomain
	RuleApplicationCacheUptime	Amount of time since rule application cache was created
	ProcessUptime	Amount of time InRule has been up in the current AppDomain
	ThreadId	Managed identification numbers for the thread that wrote the log entry
	ProcessId	Identification number for the process hosting the rules engine
	Installer Version	Version of the installer used to install InRule
	irSDK Version	Version of the InRule SDK
	HostAppDomainHeapMemoryMB	Current heap memory usage for the

		application domain associated with the running instance of InRule **Note:** This value is intermittently logged

4.6.1.1.3.2 Create Entity Event

InRule.Runtime Info Level Logging - Create Entity Event

Note: Some of the log detail items are optional and will not be present in every log entry.

Type	Name	Description
RuleSession.CreateEntity	Time	The date and time of the event
	Source	The source of the event within InRule.Runtime (e.g. "RuleSession")
	Message	The type of the event (e.g. "RuleSession.CreateEntity")
	CreateEntityTime	Elapsed time to create the entity
	RunningTotalAllTime	Elapsed time to complete the entity creation (CreateEntityTime)
	SessionId	GUID representing a unique RuleSession
	RuleApplication	Name of the rule application, and associated revision number. A revision number of "-1" indicates a file based rule application where revision number is not applicable.
	MaxRuleApplicationCacheDepth	Maximum rule apps that will remain cached before rule apps are removed
	CurrentRuleApplicationCacheDepth	Current number of rule apps in the cache
	CurrentDataQueryCacheDepth	Current number of cached queries in the AppDomain

	RuleApplicationCacheUptime	Amount of time since rule application cache was created
	ProcessUptime	Amount of time InRule has been up in the current AppDomain
	ThreadId	Managed identification numbers for the thread that wrote the log entry
	ProcessId	Identification number for the process hosting the rules engine
	Installer Version	Version of the installer used to install InRule
	irSDK Version	Version of the InRule SDK
	HostAppDomainHeapMemoryMB	Current heap memory usage for the application domain associated with the running instance of InRule **Note:** This value is intermittently logged

4.6.1.1.3.3 Apply Rules Event

InRule.Runtime Info Level Logging - Apply Rules Event

Note: Some of the log detail items are optional and will not be present in every log entry.

Type	Name	Description
RuleSession.ApplyRules	Time	The date and time of the event
	Source	The source of the event within InRule.Runtime (e.g. "RuleSession")
	Message	The type of the event (e.g. "RuleSession.ApplyRules")
	FunctionCompileTime	Elapsed time to generate and compile IL execution delegates invoked by the rule engine.

	LoadEntityXMLTime	Elapsed time to load entities from Xml. **Note:** Only logged when Xml is used to set the state of the rule application entities
	RuleExecutionTime	Elapsed time to run the rules. This includes any FunctionCompileTime, BoundStateRefreshTime, ExternalMethodCallTime and ExternalSqlQueryCallTime.
	BoundStateRefreshTime	Elapsed time to refresh state from bound object **Note:** Only logged when bound to a .NET assembly
	ExternalMethodCallTime	Elapsed time to execute User Defined Functions (UDFs) or external .NET function library methods. The number to the right of the elapsed time is the number of methods called.
	ExternalSqlQueryCallTime	Elapsed time to execute SQL queries against external data sources. The number to the right of the elapsed time indicates the number of queries executed.
	RunningTotalAllTime	Total elapsed time including compilation, entity creation and execution
	SessionId	GUID representing a unique RuleSession
	RuleApplication	Name of the rule application, and associated revision number. A revision number of "-1" indicates a file based rule application where revision number is not applicable.
	MaxRuleApplicationCacheDepth	Maximum rule apps that will remain cached before rule apps are removed
	CurrentRuleApplicationCacheDepth	Current number of rule apps in the cache

	CurrentDataQueryCacheDepth	Current number of cached queries in the AppDomain
	RuleApplicationCacheUptime	Amount of time since rule application cache was created
	ProcessUptime	Amount of time InRule has been up in the current AppDomain
	ThreadId	Managed identification numbers for the thread that wrote the log entry
	ProcessId	Identification number for the process hosting the rules engine
	Installer Version	Version of the installer used to install InRule
	irSDK Version	Version of the InRule SDK
	HostAppDomainHeapMemoryMB	Current heap memory usage for the application domain associated with the running instance of InRule **Note:** This value is intermittently logged

4.6.1.1.3.4 Execute Rule Set Event

InRule.Runtime Info Level Logging - Execute Rule Set Event

Note: Some of the log detail items are optional and will not be present in every log entry.

Type	Name	Description
RuleSession.ExecuteRuleSet	Time	The date and time of the event
	Source	The source of the event within InRule.Runtime (e.g. "RuleSession")
	Message	The type of the event (e.g. "RuleSession.ExecuteRuleSet") This event type will include the name of the rule set executed.
	FunctionCompileTime	Elapsed time to generate and compile

		IL execution delegates invoked by the rule engine.
	LoadEntityXMLTime	Elapsed time to load entities from Xml. **Note:** Only logged when Xml is used to set the state of the rule application entities
	RuleExecutionTime	Elapsed time to run the rules. This includes any FunctionCompileTime, BoundStateRefreshTime, ExternalMethodCallTime and ExternalSqlQueryCallTime.
	BoundStateRefreshTime	Elapsed time to refresh state from bound object **Note:** Only logged when bound to a .NET assembly
	ExternalMethodCallTime	Elapsed time to execute User Defined Functions (UDFs) or external .NET function library methods. The number to the right of the elapsed time is the number of methods called.
	ExternalSqlQueryCallTime	Elapsed time to execute SQL queries against external data sources. The number to the right of the elapsed time indicates the number of queries executed.
	RunningTotalAllTime	Total elapsed time including compilation, entity creation and execution
	SessionId	GUID representing a unique RuleSession
	RuleApplication	Name of the rule application, and associated revision number. A revision number of "-1" indicates a file based rule application where revision number is not applicable.
	MaxRuleApplicationCacheDepth	Maximum rule apps that will remain cached before rule apps are

		removed
	CurrentRuleApplicationCacheDepth	Current number of rule apps in the cache
	CurrentDataQueryCacheDepth	Current number of cached queries in the AppDomain
	RuleApplicationCacheUptime	Amount of time since rule application cache was created
	ProcessUptime	Amount of time InRule has been up in the current AppDomain
	ThreadId	Managed identification numbers for the thread that wrote the log entry
	ProcessId	Identification number for the process hosting the rules engine
	Installer Version	Version of the installer used to install InRule
	irSDK Version	Version of the InRule SDK
	HostAppDomainHeapMemoryMB	Current heap memory usage for the application domain associated with the running instance of InRule **Note:** This value is intermittently logged

4.6.1.2 Repository Event Log Details

InRule.Repository Warn Level Logging

Type	Name	Description
Repository warning	Message	Warning description
	ThreadId	Thread ID where the rule engine process is running
	HostAppDomainHeapMemoryMB	Amount of heap memory used by the host application in MB
	Error Information	Exception details

InRule.Repository Info Level Logging

Type	Name	Description
GetRuleAppSummary	RepositoryServiceClientAggExecStats.StartTimeStamp	

	RepositoryServiceClientAggExecStats.GetTime	
	RepositoryServiceClientAggExecStats.OtherServiceCallTime	
	RepositoryServiceClientAggExecStats.GetInfoTime	
	RepositoryServiceClientAggExecStats.DefSerializationTime	
	ThreadId	
	RepositoryServiceClientAggExecStats.RunningTotalAll	
	HostAppDomainHeapMemoryMB	
GetDefsForRuleApp	RepositoryServiceClientAggExecStats.StartTimeStamp	
	RepositoryServiceClientAggExecStats.OtherServiceCallTime	
	RepositoryServiceClientAggExecStats.GetInfoTime	
	RepositoryServiceClientAggExecStats.DefSerializationTime	
	ThreadId	
	RepositoryServiceClientAggExecStats.RunningTotalAll	
CreateRuleApplication	RepositoryServiceClientAggExecStats.StartTimeStamp	
	RepositoryServiceClientAggExecStats.OtherServiceCallTime	
	RepositoryServiceClientAggExecStats.DefSerializationTime	
	ThreadId	
	RepositoryServiceClientAggExecStats.RunningTotalAll	
	HostAppDomainHeapMemoryMB	
UndoRuleAppCheckout	RepositoryServiceClientAggExecStats.StartTimeStamp	
	RepositoryServiceClientAggExecStats.GetTime	
	RepositoryServiceClientAggExecStats.OtherServiceCallTime	
	RepositoryServiceClientAggExecStats.GetInfoTime	
	RepositoryServiceClientAggExecStats.DefSerializationTime	
	ThreadId	
	RepositoryServiceClientAggExecStats.RunningTotalAll	
	HostAppDomainHeapMemoryMB	

GetCheckoutSets	RepositoryServiceClientAggExecStats.StartTimeStamp	
	RepositoryServiceClientAggExecStats.GetTime	
	RepositoryServiceClientAggExecStats.OtherServiceCallTime:	
	RepositoryServiceClientAggExecStats.GetInfoTime	
	RepositoryServiceClientAggExecStats.DefSerializationTime	
	RepositoryServiceClientAggExecStats.RunningTotalAll	
	HostAppDomainHeapMemoryMB	
CheckoutRuleApplication	RepositoryServiceClientAggExecStats.StartTimeStamp	
	RepositoryServiceClientAggExecStats.GetTime	
	RepositoryServiceClientAggExecStats.OtherServiceCallTime	
	RepositoryServiceClientAggExecStats.GetInfoTime	
	RepositoryServiceClientAggExecStats.DefSerializationTime	
	ThreadId	
	RepositoryServiceClientAggExecStats.RunningTotalAll	
	HostAppDomainHeapMemoryMB	
GetStaleDefsForRuleApp	RepositoryServiceClientAggExecStats.StartTimeStamp	
	RepositoryServiceClientAggExecStats.GetTime	
	RepositoryServiceClientAggExecStats.OtherServiceCallTime	
	RepositoryServiceClientAggExecStats.GetInfoTime	
	RepositoryServiceClientAggExecStats.DefSerializationTime	
	ThreadId	
	RepositoryServiceClientAggExecStats.RunningTotalAll	
	HostAppDomainHeapMemoryMB	
CheckIn	RepositoryServiceClientAggExecStats.StartTimeStamp	
	RepositoryServiceClientAggExecStats.GetTime	
	RepositoryServiceClientAggExecStats.OtherServiceCallTime	

	RepositoryServiceClientAggExecStats.GetInfoTime	
	RepositoryServiceClientAggExecStats.DefSerializationTime	
	ThreadId	
	RepositoryServiceClientAggExecStats.RunningTotalAll	
GetLatestRuleAppRevision	RepositoryServiceClientAggExecStats.StartTimeStamp	
	RepositoryServiceClientAggExecStats.GetTime	
	RepositoryServiceClientAggExecStats.OtherServiceCallTime	
	RepositoryServiceClientAggExecStats.GetInfoTime	
	RepositoryServiceClientAggExecStats.DefSerializationTime	
	ThreadId	
	RepositoryServiceClientAggExecStats.RunningTotalAll	
CheckoutDef	RepositoryServiceClientAggExecStats.StartTimeStamp	
	RepositoryServiceClientAggExecStats.GetTime	
	RepositoryServiceClientAggExecStats.OtherServiceCallTime	
	RepositoryServiceClientAggExecStats.GetInfoTime	
	RepositoryServiceClientAggExecStats.DefSerializationTime	
	ThreadId	
	RepositoryServiceClientAggExecStats.RunningTotalAll	

4.6.2 Performance Statistics Report

Overview

The performance detail report provides detailed execution information that can assist in the performance analysis of a rule application. Statistics are captured for the following levels of the rule application:

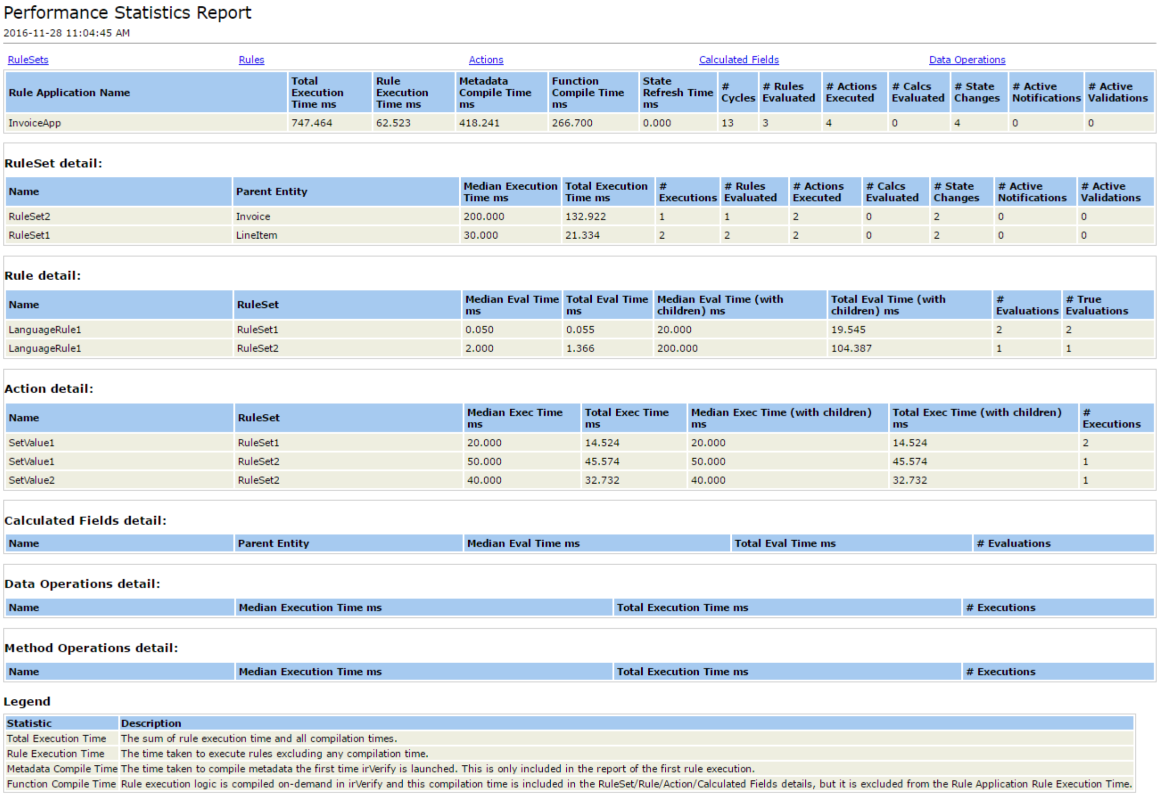
- Rule application
- RuleSets
- Rules
- Actions
- Calculations
- Data Operations

Information captured:

- Execution times (Total and Mean)
- Number of Executions, Cycles and Rule Evaluations

Availability

The report is accessible in irVerify by clicking the Performance Statistics button or via the SDK as shown in the Source Code Sample "Retrieve the Performance Statistics Report."



Performance Statistics Report

2016-11-28 11:04:45 AM

RuleSets Rules Actions Calculated Fields Data Operations

Rule Application Name	Total Execution Time ms	Rule Execution Time ms	Metadata Compile Time ms	Function Compile Time ms	State Refresh Time ms	# Cycles	# Rules Evaluated	# Actions Executed	# Calcs Evaluated	# State Changes	# Active Notifications	# Active Validations
InvoiceApp	747.464	62.523	418.241	266.700	0.000	13	3	4	0	4	0	0

RuleSet detail:

Name	Parent Entity	Median Execution Time ms	Total Execution Time ms	# Executions	# Rules Evaluated	# Actions Executed	# Calcs Evaluated	# State Changes	# Active Notifications	# Active Validations
RuleSet2	Invoice	200.000	132.922	1	1	2	0	2	0	0
RuleSet1	LineItem	30.000	21.334	2	2	2	0	2	0	0

Rule detail:

Name	RuleSet	Median Eval Time ms	Total Eval Time ms	Median Eval Time (with children) ms	Total Eval Time (with children) ms	# Evaluations	# True Evaluations
LanguageRule1	RuleSet1	0.050	0.055	20.000	19.545	2	2
LanguageRule1	RuleSet2	2.000	1.366	200.000	104.387	1	1

Action detail:

Name	RuleSet	Median Exec Time ms	Total Exec Time ms	Median Exec Time (with children) ms	Total Exec Time (with children) ms	# Executions
SetValue1	RuleSet1	20.000	14.524	20.000	14.524	2
SetValue1	RuleSet2	50.000	45.574	50.000	45.574	1
SetValue2	RuleSet2	40.000	32.732	40.000	32.732	1

Calculated Fields detail:

Name	Parent Entity	Median Eval Time ms	Total Eval Time ms	# Evaluations

Data Operations detail:

Name	Median Execution Time ms	Total Execution Time ms	# Executions

Method Operations detail:

Name	Median Execution Time ms	Total Execution Time ms	# Executions

Legend

Statistic	Description
Total Execution Time	The sum of rule execution time and all compilation times.
Rule Execution Time	The time taken to execute rules excluding any compilation time.
Metadata Compile Time	The time taken to compile metadata the first time irVerify is launched. This is only included in the report of the first rule execution.
Function Compile Time	Rule execution logic is compiled on-demand in irVerify and this compilation time is included in the RuleSet/Rule/Action/Calculated Fields details, but it is excluded from the Rule Application Rule Execution Time.

4.7 irServer - Rule Execution Service

.NET applications or third-party software products (e.g. BPM products) can call the Rules Engine via the Rule Execution Service without needing to reference irSDK.
The Rule Execution Service currently supports the REST and SOAP protocols.

See also InRule Runtime Service Config File Settings section for information on storing irCatalog credentials for use by irServer's REST endpoint, and configuring Endpoint Overrides in the config file.

The following topics are provided for assistance with the setup, configuration, and use of the Rule Execution Service:

- Accessing the Rule Execution Service via SOAP
  - Adding a Service Reference in Visual Studio
  - Calling irServer SOAP endpoint using a Service Reference

  - Executing a Decision on irServer SOAP endpoint using a Service Reference
  - Configuring irServer SOAP Endpoint to support WsHttpBinding
  - Handling irServer SOAP Endpoint Error Conditions
  - Caching Behavior

- Accessing the Rule Execution Service via REST
  - Methods
    - Apply Rules Sample Request and Response Formats
    - Execute Decision Sample Request and Response Formats
    - Execute Independent RuleSet Sample Request and Response Formats
    - Execute RuleSet Sample Request and Response Formats
  - HTTP Request Member Definitions
    - RuleApp
      - RepositoryRuleAppRevisionSpec
    - RuleEngineServiceOptions
    - RuleEngineServiceOutputTypes
  - Overriding RuleApp Endpoints at Runtime
    - Database Connection String
    - Mail Server Connection
    - Web Service Address
    - Web Service WSDL Uri
    - XML Document Path
    - XML Schema
    - XML Schema Validation
    - Inline Table
    - Inline XML Document
    - Inline Value List
    - SQL Query
    - REST Service X.509 Certificate Path
    - REST Service Authentication Type
    - REST Service Root Url

4.7.1 Accessing via SOAP

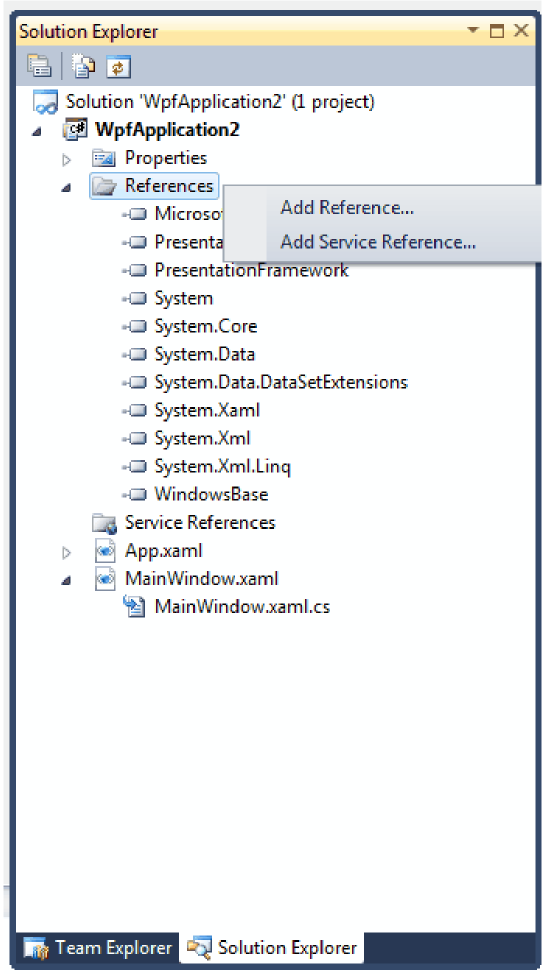
The following topics and samples are provided for assistance with the use of the SOAP Endpoint:

- Adding a Service Reference in Visual Studio
- Calling irServer SOAP endpoint using a Service Reference
- Executing a Decision on irServer SOAP endpoint using a Service Reference
- Configuring irServer SOAP Endpoint to support WsHttpBinding
- Handling irServer SOAP Endpoint Error Conditions
- Caching Behavior

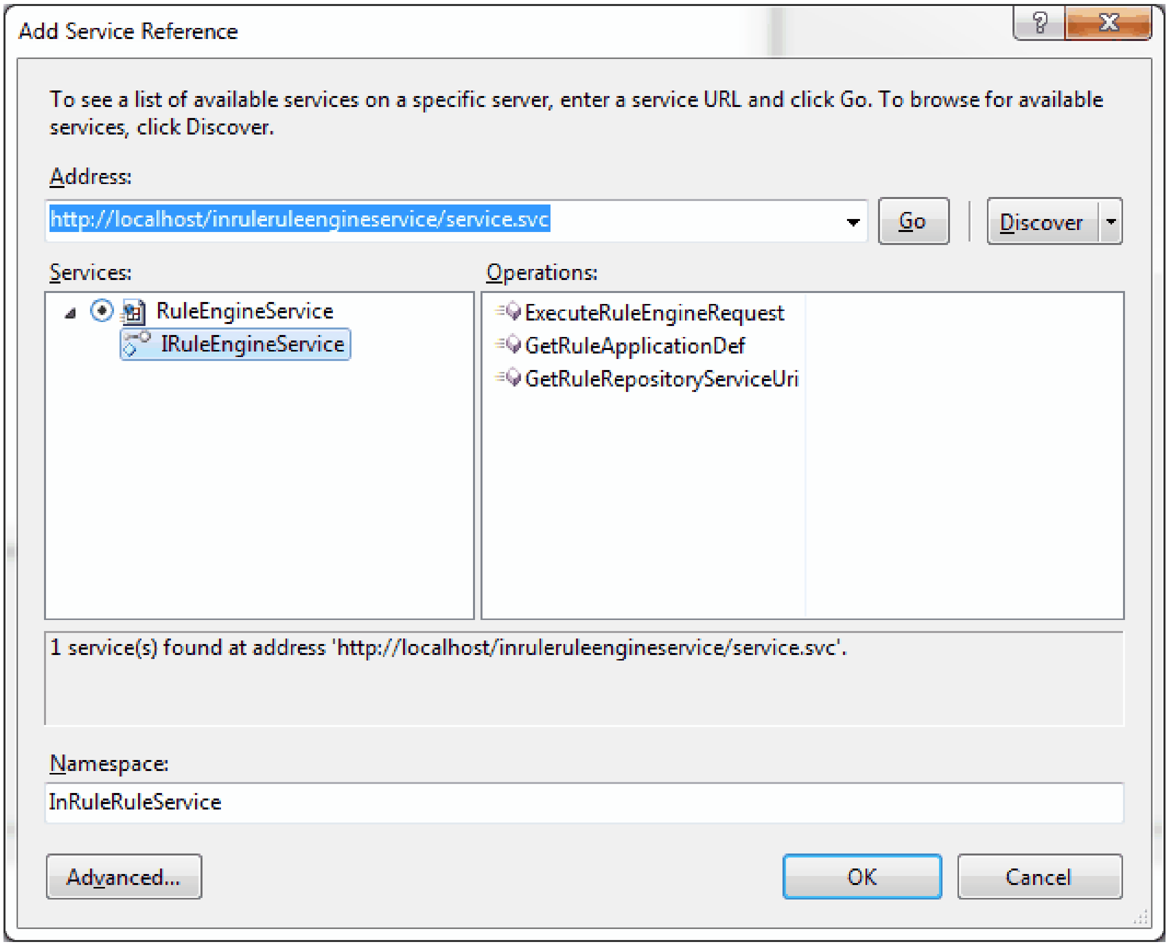
4.7.1.1 Adding a Service Reference in Visual Studio

To configure a Service Reference, do the following:

1. Right click on the References folder in the project. From the menu select **Add Service Reference**.



2. You will be prompted with the Add Service Reference dialog. Enter the URL of the InRule Rule Service (e.g. http://server/InRuleRuleEngineService/service.svc)



To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

Address: http://localhost/inruleruleengineservice/service.svc

Services:

- RuleEngineService
  - IRuleEngineService

Operations:

- ExecuteRuleEngineRequest
- GetRuleApplicationDef
- GetRuleRepositoryServiceUri

1 service(s) found at address 'http://localhost/inruleruleengineservice/service.svc'.

Namespace: InRuleRuleService

3. Change the Namespace field to something descriptive such as RuleEngineService.
4. Click OK button. This will add the Service Reference to your project.
5. Follow the code sample to Calling irServer SOAP Endpoint using a Service Reference.

4.7.1.2 Calling irServer SOAP Endpoint using a Service Reference

Prerequisites: A valid Service Reference
See Also: irServer - Rule Execution Service

The following sample represents how to call irServer Rule Execution Service using a service reference:

```
using (RuleEngineServiceClient proxy = new RuleEngineServiceClient())
{
    try
    {
            // Get RuleApp as defined in the config (RepositoryRuleApp or
FileSystemRuleApp)
            RepositoryRuleApp rules = new RepositoryRuleApp();
            rules.RepositoryServiceUri = "http://server/InRuleCatalogService/
Service.svc";
            RepositoryRuleAppRevisionSpec spec = new
```

```
RepositoryRuleAppRevisionSpec();
          spec.RuleApplicationName = "MortgageCalculator";
          rules.RepositoryRuleAppRevisionSpec = spec;
          rules.UserName = "Admin";
          rules.Password = "password";

          // Create new ApplyRulesRequest
          ApplyRulesRequest request = new ApplyRulesRequest();
          request.RuleApp = rules;
          request.EntityName = "Mortgage";
          request.RuleEngineServiceOutputTypes = new
RuleEngineServiceOutputTypes();
          request.RuleEngineServiceOutputTypes.ActiveNotifications = true;
          request.RuleEngineServiceOutputTypes.ActiveValidations = true;
          request.RuleEngineServiceOutputTypes.EntityState = true;

          // Load state XML
          Console.WriteLine("- Loading XML state for 'Invoice'...");
          request.EntityState = "<Mortgage><LoanInfo><Principal>500000</
Principal><APR>6.875</APR>< TermInYears > 30 </TermInYears ></LoanInfo
><PaymentSummary/ ></Mortgage > ";
          Console.WriteLine("Input State:");
          Console.WriteLine(request.EntityState);
          Console.WriteLine("");

          // Submit Request
          Console.WriteLine("- Calling ApplyRules() from RuleEngineService...");
          RuleEngineServiceResponse response =
proxy.ExecuteRuleEngineRequest(request);

          Console.WriteLine("Active Notifications:");
          foreach (Notification notification in response.ActiveNotifications)
          {
                 Console.WriteLine(notification.NotificationType + ": " +
notification.Message);
          }
          Console.WriteLine("");

          Console.WriteLine("Active Validations:");
          foreach (Validation validation in response.ActiveValidations)
          {
                 Console.WriteLine(validation.InvalidMessageText);
          }
          Console.WriteLine("");

          Console.WriteLine("Output State:");
          // Note: XML formatting not maintained in response
          Console.WriteLine(response.EntityState);
          Console.WriteLine("");
   }
   catch (Exception ex)
   {
          Console.WriteLine("Unknown exception occurred during RuleEngineService
request: " + ex.ToString());
```

```
    }
}

Console.WriteLine("[END ServiceReferenceConsumer Sample]");
```

4.7.1.3 Executing a Decision on irServer SOAP endpoint using a Service Reference

Prerequisites: A valid Service Reference
See Also: irServer - Rule Execution Service

The following sample represents how to execute a Decision on irServer Rule Execution Service using a service reference:

```
using (RuleEngineServiceClient proxy = new RuleEngineServiceClient())
{
    // Get RuleApp as defined in the config (RepositoryRuleApp or
FileSystemRuleApp)
    RepositoryRuleApp ruleApp = new RepositoryRuleApp
    {
          RepositoryServiceUri = "http://localhost/InRuleCatalogService/
Service.svc",
          RepositoryRuleAppRevisionSpec = new RepositoryRuleAppRevisionSpec
{ RuleApplicationName = "AreaCalculator" },
          UserName = "Admin",
          Password = "password"
    };

    // Create new ExecuteDecisionRequest
    ExecuteDecisionRequest request = new ExecuteDecisionRequest
    {
          RuleApp = ruleApp,
          DecisionName = "CalculateArea",
          InputState = "{\"Height\":50,\"Width\":3}",     // Note: Decisions only
accept JSON input
          RuleEngineServiceOutputTypes = new RuleEngineServiceOutputTypes
                                           {
                                                  ActiveNotifications = true,
                                                  ActiveValidations = true
                                           }
    };

    try
    {
          // Submit request
          Console.WriteLine("- Calling ExecuteDecition on RuleEngineService...");
          ExecuteDecisionResponse response = (ExecuteDecisionResponse)
proxy.ExecuteRuleEngineRequest(request);

          Console.WriteLine("Active Notifications:");
          foreach (Notification notification in response.ActiveNotifications)
          {
                 Console.WriteLine($"[{notification.NotificationType}]:
{notification.Message}");
```

```
           }
           Console.WriteLine("");

           Console.WriteLine("Active Validations:");
           foreach (Validation validation in response.ActiveValidations)
           {
                  Console.WriteLine(validation.InvalidMessageText);
           }
           Console.WriteLine("");

           // Note: Decisions only support the output of JSON, not XML
           Console.WriteLine("Decision Output:");
           Console.WriteLine(response.OutputState);
           Console.WriteLine("");
    }
    catch (Exception ex)
    {
           Console.WriteLine($"Unknown exception occurred during RuleEngineService
request: {ex}");
    }
}
Console.WriteLine("[END ExecuteDecisionServiceReferenceConsumer Sample]");
```

4.7.1.4 Configuring irServer SOAP Endpoint to Support WsHttpBinding

Bindings for irServer are driven by the configuration file. Therefore, to provide support for WsHttpBinding one simply needs to modify the service configuration file (web.config in the case of IIS hosted service or InRuleRuntimeService.config in the case of Windows Service) and the client configuration file (in the case of a .NET client).

Refer to .NET Framework documentation for further information about application settings, which can be found at http://msdn.microsoft.com/en-us/library/ms731734.aspx.

Note: When running the Rule Engine Service hosted under IIS, you can only configure one protocol in the web.config file. This does not apply if the Rule Engine service runs as a Windows Service.

Service Configuration Snippet

```
<system.serviceModel>
    <behaviors>
      <serviceBehaviors>
           <behavior name="Behavior_IRuleEngineService">
             <serviceDebug httpHelpPageEnabled="true" />
             <serviceMetadata httpGetEnabled="true" />
           </behavior>
      </serviceBehaviors>
    </behaviors>
    <bindings>
      <wsHttpBinding>
           <binding name="WsHttpBinding_IRuleEngineService"
maxReceivedMessageSize="2147483647">
             <readerQuotas maxStringContentLength="2147483647"
maxArrayLength="2147483647" />
             <security mode="None" />
           </binding>
      </wsHttpBinding>
    </bindings>
    <services>
```

```
      <service name="InRule.Runtime.Service.RuleEngineService"
behaviorConfiguration="Behavior_IRuleEngineService">
           <host>
             <baseAddresses>
                  <add baseAddress="http://localhost:8083/InRuleRuleEngineService-
WsHttpBinding/Service.svc" />
             </baseAddresses>
           </host>
           <endpoint address="http://localhost:8083/InRuleRuleEngineService-
WsHttpBinding/Service.svc"
                     contract="InRule.Runtime.Service.IRuleEngineService"
                     binding="wsHttpBinding"
                     bindingConfiguration="WsHttpBinding_IRuleEngineService" />
      </service>
    </services>
</system.serviceModel>
```

Client Configuration Snippet

```
<system.serviceModel>
    <client>
      <endpoint address="http://localhost:8083/InRuleRuleEngineService-
WsHttpBinding/Service.svc"
                contract="InRule.Runtime.Service.IRuleEngineService"
                binding="wsHttpBinding"
                bindingConfiguration="WsHttpBinding_IRuleEngineService">
      </endpoint>
    </client>
    <bindings>
      <wsHttpBinding>
           <binding name="WsHttpBinding_IRuleEngineService"
maxReceivedMessageSize="2147483647">
             <readerQuotas maxStringContentLength="2147483647"
maxArrayLength="2147483647" />
             <security mode="None" />
           </binding>
      </wsHttpBinding>
    </bindings>
</system.serviceModel>
```

4.7.1.5 Handling irServer SOAP Endpoint Error Conditions

irServer Rule Execution Service SOAP Endpoint uses different error reporting mechanisms depending on the type of error encountered on the server.

Normally any error conditions or exceptions encountered on the server are reported to the client via the WCF Fault mechanism. These will likely occur during RuleApplication compilation or erroneous state modifications. Runtime errors, however, are generally large in size due to the embedded RuleExecutionLog, RuleSessionState and AggExecStats; these return to the client as a regular service response, but the response is flagged as having runtime errors.

When using a ServiceReference proxy client, you should always look for runtime errors. Check the `RuleEngineServiceResponse.HasRuntimeErrors` property after each request to determine whether the attached RuleExecutionLog contains errors.

The following code snippets from the irServer Rule Execution Service Samples illustrate how the different error conditions can be detected:

Creating an Invalid Entity

```
try
{
    // Ask the RuleEngineService for its associated Repository Service URI
    Console.WriteLine("- Requesting RespitoryServiceUri from
RuleEngineService...");
    string repositoryServiceUri = proxy.GetRuleRepositoryServiceUri();
    if (String.IsNullOrEmpty(repositoryServiceUri))
    {
           throw (new Exception("RuleEngineService is not configured to use a
specific Repository Service."));
    }

    // Get RuleApp as defined in the config (RepositoryRuleApp or
FileSystemRuleApp)
    RuleApp ruleApp = GetRuleApp(repositoryServiceUri);

    // Construct a request that will fail with an Exception
    ApplyRulesRequest request = new ApplyRulesRequest();
    request.RuleApp = ruleApp;
    request.EntityName = "InvalidEntityName123"; // Use an invalid Entity name
    request.RuleEngineServiceOutputTypes = new RuleEngineServiceOutputTypes();
    request.RuleEngineServiceOutputTypes.ActiveNotifications = true;
    request.RuleEngineServiceOutputTypes.ActiveValidations = true;
    request.RuleEngineServiceOutputTypes.EntityState = true;
    request.EntityState = "<InvalidEntityName123/>";
    Console.WriteLine("- Calling ApplyRules() from RuleEngineService...");
    Console.WriteLine("");
    proxy.ExecuteRuleEngineRequest(request); // This should throw a contracted
Fault
}
catch (FaultException<ServiceFault> contractedFault)
{
    Console.WriteLine("Received expected contracted fault:");
    Console.WriteLine("Exception Type:" + contractedFault.Detail.QualifiedName);
    Console.WriteLine("Exception Message:" + contractedFault.Detail.Message);
}
```

Handling a Runtime Error

```
// Get RuleApp designed to fail at Runtime
RuleApp ruleApp = GetDivideByZeroRuleApp();

// Construct a request that will fail *without* throwing an Exception
ApplyRulesRequest request = new ApplyRulesRequest();
request.RuleApp = ruleApp;
request.EntityName = "Entity1"; // Use an invalid Entity name
request.RuleEngineServiceOutputTypes = new RuleEngineServiceOutputTypes();
request.RuleEngineServiceOutputTypes.EntityState = true;
request.EntityState = "<Entity1/>";
Console.WriteLine("- Calling ApplyRules() from RuleEngineService...");
Console.WriteLine("");
```

```
// This should succeed, but the response should have a RuntimeError flag set to
true
RuleEngineServiceResponse response = proxy.ExecuteRuleEngineRequest(request);

if (response.HasRuntimeErrors)
{
    // Iterate all the Runtime errors in the Execution Log
    foreach (RuleExecutionLogMessage msg in response.RuleExecutionLog.Messages)
    {
            if (msg is ErrorMessage)
            {
                    ErrorMessage error = (ErrorMessage)msg;
                    Console.WriteLine("Expected RuntimeError: " + error.Message);
            }
    }
}
```

4.7.1.6 Caching Behavior

irServer Rule Execution Service RuleApp Cache Characteristics and Significance

When a rule application is submitted for first time execution, the rule application is compiled and then cached in the AppDomain. Subsequent requests for the rule application will pull from the AppDomain, providing faster access to the rule application. To compile the rule application and enable caching it should be requested appropriately. The compile and caching process is sometimes referred to as the cold start. See Cold Start Mitigation for techniques to avoid end users experiencing time lags due to this behavior.

The FileSystemRuleApp, RepositoryRuleApp and InMemoryRuleApp all leverage caching.

Each rule application type will check to see if there have been changes made to the rule application (or if there is a new revision of the rule application in the case of the RepositoryRuleApp) and will recompile and cache the rule application if there are changes.

By default, 5 rule applications will be stored in the cache. To modify the cache limit, see Managing the RuleApp Cache.

Any time the AppDomain shuts down (e.g. IISReset or AppPool settings such as Recycling or Performance settings), the cache is cleared.

Catalog Specific Caching Behavior

- When a rule application is pulled from the Catalog, it is cached on the Rule Execution Service side, not the Catalog Service.
- If the RuleApp is in cache, and within the specified cache timeout, the cached RuleApp is used. The default cache timeout is 30 seconds. The timeout can be overridden by passing the desired timeout value into the RepositoryRuleApp constructor.
- If the cache has expired, a lightweight check is performed to see if a newer revision of the RuleApp exists using GetRuleApplicationDef(). If a new revision does not exist, the cached RuleApp is used.
- If the RuleApp is not in the cache, it is pulled from the Catalog.

Requesting irServer Rule Execution Service to use the specific revision of the rule application from catalog can be performed using the following RepositoryRuleAppRevisionSpec class:

```
var spec = new RepositoryRuleAppRevisionSpec { RuleApplicationName = "IrSoaTest",
Label = "", Revision = 1 };
    ruleApp.RepositoryRuleAppRevisionSpec = spec;
```

InMemory Specific Caching Behavior

To enable caching behavior for InMemoryRuleApp, the following steps are required:

- Submitting a request to obtain the caching key will retrieve the key which uniquely identifies ruleapp on the irServer Rule Execution Service:

```
var ruleApp = new InMemoryRuleApp();
ruleApp.RuleApplicationDef = File.ReadAllText(ruleAppFilename);

var request = new RuleApplicationInfoRequest();
request.RuleApp = ruleApp;
var proxy = new RuleEngineServiceClient();

var ruleAppInfo  = proxy.GetRuleApplicationInfo(request);
```

ruleAppInfo.RuleApplicationTrackingKey contains the caching key, which is used for the subsequent requests.

- Submitting a request to execute a rule application using a cached rule application key retrieved in the prior step will use the cached revision:

```
var ruleApp = new InMemoryRuleApp();
ruleApp.CacheAndRevisionKey = ruleAppInfo.RuleApplicationTrackingKey;
var request = new ApplyRulesRequest();
request.RuleApp = ruleApp;
```

***Note:** complete ruleapplication xml in each subsequent request is not required for InMemoryRuleApp as provided caching key directing irServer Rule Execution Service to lookup the cache for the specific revision.*

FileSystem Caching Specific Behavior

- When a rule application is pulled from the File System, it is cached on the server. For each request, file on the disk is checked if it was updated and reloaded if so.
- Path to the rule application can be specified using the FilePath property of the FileSystemRuleApp class:

```
var ruleApp = new FileSystemRuleApp();
ruleApp.FilePath = “filepath”;
```

4.7.2 Accessing via REST

irServer Rule Execution Service supports use of standard HTTP verbs to execute rules in addition to the existing SOAP Endpoint support.

The irServer Rule Execution Service uses content negotiation to determine the request and response types. **Note:** If not specified, both the request and response types will default to XML.

- Request Examples

    - Content-Type: application/xml
    - Content-Type: application/json
- Response Examples
    - Accept: application/xml
    - Accept: application/json

The following topics and samples are provided for assistance with the use of the REST Endpoint:

- Methods
    - Apply Rules Sample Request and Response Formats
    - Execute Decision Sample Request and Response Formats
    - Execute Independent RuleSet Sample Request and Response Formats
    - Execute RuleSet Sample Request and Response Formats
- HTTP Request Member Definitions
    - RuleApp
        - RepositoryRuleAppRevisionSpec
    - RuleEngineServiceOptions
    - RuleEngineServiceOutputTypes
- Overriding RuleApp Endpoints at Runtime
    - Database Connection String
    - Mail Server Connection
    - Web Service Address
    - Web Service WSDL Uri
    - XML Document Path
    - XML Schema
    - XML Schema Validation
    - Inline Table
    - Inline XML Document
    - Inline Value List
    - SQL Query
    - REST Service X.509 Certificate Path
    - REST Service Authentication Type
    - REST Service Root Url

See also InRule Runtime Service Config File Settings section for information on storing irCatalog credentials in the config file for use by irServer's REST endpoint.

4.7.2.1 Methods

The following topics and samples are provided for assistance with the use of the REST Endpoint methods:

- Apply Rules Sample Request and Response Formats
- Execute Decision Sample Request and Response Formats
- Execute Independent RuleSet Sample Request and Response Formats
- Execute RuleSet Sample Request and Response Formats

4.7.2.1.1 Apply Rules

URL:	e.g. http://servername/InRuleRuleEngineService/HttpService.svc/ApplyRules
HTTP Method:	POST

See below for examples of requests / responses in both XML and JSON formats. The variations shown are:

- File-based Rule Application (deployed on irServer)
- Catalog Rule Application
  - Rule Application specified by either Name or Guid
  - Revision specified as Latest, Label or Revision
  - Authentication using specified Username / Password, Single Sign-On or using irServer config-specified credentials
- Optional RequestId specified
- Optional CacheTimeout specified
- Optional ConnTimeout specified
- Optional Overrides specified
- Optional Output Options specified (overriding defaults)

Note: In all XML request examples below, elements must be ordered exactly as specified.

Sample XML Request: Filesystem Rule Application

```
<ApplyRulesRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <FileName>Ruleapp1.ruleapp</FileName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
</ApplyRulesRequest>
```

Sample XML Request: Catalog Rule Application: Latest Revision by Name

```
<ApplyRulesRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
      <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
</ApplyRulesRequest>
```

Sample XML Request: Catalog Rule Application: Latest Revision by Guid

```
<ApplyRulesRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
      <Guid>5f11d845-bda1-4f79-ba67-e8cd12d065f7</Guid>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
</ApplyRulesRequest>
```

Sample XML Request: Catalog Rule Application: Labeled Revision by Name

```
<ApplyRulesRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
      <Label>Label1</Label>
      <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
</ApplyRulesRequest>
```

Sample XML Request: Catalog Rule Application: Specific Revision by Name

```
<ApplyRulesRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
       <Revision>3</Revision>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
</ApplyRulesRequest>
```

Sample XML Request: Catalog Rule Application: SSO Authentication (using service host account credential)

```
<ApplyRulesRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UseIntegratedSecurity>true</UseIntegratedSecurity>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
</ApplyRulesRequest>
```

Sample XML Request: Catalog Rule Application: irServer config-specified credentials Authentication

```
<ApplyRulesRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
</ApplyRulesRequest>
```

Sample XML Request: Catalog Rule Application: RequestId, CacheTimeout, ConnTimeout, Overrides, Output Options specified

```
<ApplyRulesRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RequestId>1627aea5-8e0a-4371-9022-9b504344e724</RequestId>
  <RuleApp>
    <CacheTimeout>100</CacheTimeout>
    <ConnTimeout>120</ConnTimeout>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <!-- see Overrides for examples -->
      </Override>
    </Overrides>
    <RuleSessionOverrides>
      <ExecutionTimeout>PT40S</ExecutionTimeout>
      <MaxCycleCount>200000</MaxCycleCount>
      <Now>1999-05-31T11:20:00</Now>
    </RuleSessionOverrides>
  </RuleEngineServiceOptions>
  <RuleEngineServiceOutputTypes>
    <ActiveNotifications>true</ActiveNotifications>
    <ActiveValidations>true</ActiveValidations>
    <EntityState>true</EntityState>
    <Overrides>true</Overrides>
    <RuleExecutionLog>true</RuleExecutionLog>
  </RuleEngineServiceOutputTypes>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
</ApplyRulesRequest>
```

Sample JSON Request: Filesystem Rule Application

```
{
   "RuleApp": {
      "FileName": "Ruleapp1.ruleapp"
   },
   "EntityName":"Entity1",
   "EntityState":"{\"Field1\":\"1\"}"
}
```

Sample JSON Request: Catalog Rule Application: Latest Revision by Name

```
{
   "RuleApp":{
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UserName":"CatUsername"
   },
   "EntityName":"Entity1",
   "EntityState":"{\"Field1\":\"1\"}"
}
```

Sample JSON Request: Catalog Rule Application: Latest Revision by Guid

```
{
   "RuleApp":{
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "Guid":"5f11d845-bda1-4f79-ba67-e8cd12d065f7"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UserName":"CatUsername"
   },
   "EntityName":"Entity1",
   "EntityState":"{\"Field1\":\"1\"}"
}
```

Sample JSON Request: Catalog Rule Application: Labeled Revision by Name

```
{
   "RuleApp":{
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "Label":"Label1",
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UserName":"CatUsername"
   },
   "EntityName":"Entity1",
   "EntityState":"{\"Field1\":\"1\"}"
}
```

Sample JSON Request: Catalog Rule Application: Specific Revision by Name

```
{
   "RuleApp":{
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "Revision":3,
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UserName":"CatUsername"
   },
   "EntityName":"Entity1",
   "EntityState":"{\"Field1\":\"1\"}"
}
```

Sample JSON Request: Catalog Rule Application: SSO Authentication (using service host account credential)

```
{
   "RuleApp":{
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UseIntegratedSecurity":true
   },
   "EntityName":"Entity1",
   "EntityState":"{\"Field1\":\"1\"}"
}
```

Sample JSON Request: Catalog Rule Application: irServer config-specified credentials Authentication

```
{
   "RuleApp":{
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc"
   },
   "EntityName":"Entity1",
   "EntityState":"{\"Field1\":\"1\"}"
}
```

Sample JSON Request: Catalog Rule Application: RequestId, CacheTimeout, ConnTimeout, Overrides, Output Options specified

```
{
   "RequestId":"1627aea5-8e0a-4371-9022-9b504344e724",
   "RuleApp":{
      "CacheTimeout":"120",
      "ConnTimeout":"100",
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UseIntegratedSecurity":true,
      "UserName":"CatPassword"
   },
   "RuleEngineServiceOptions":{
      "Overrides":[
         // see Overrides for example.
      ],
      "RuleSessionOverrides":{
         "ExecutionTimeout":"PT40S",
         "MaxCycleCount":200000,
         "Now":"\/Date(928167600000-0500)\/",
      },
   },
   "RuleEngineServiceOutputTypes":{
      "ActiveNotifications":true,
      "ActiveValidations":true,
      "EntityState":true,
      "Overrides":true,
      "RuleExecutionLog":true
   },
   "EntityName":"Entity1",
   "EntityState":"{\"Field1\":\"1\"}"
}
```

Sample XML Response

```xml
<RuleEngineHttpServiceResponse xmlns="http://www.inrule.com/XmlSchema/Schema">
  <ActiveNotifications>
    <Notification>
      <Changed>true</Changed>
      <ElementId>String content</ElementId>
      <FiredBy>
        <string xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">String content</
        <string xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">String content</
      </FiredBy>
      <IsActive>true</IsActive>
      <ManagedByRuleEngine>true</ManagedByRuleEngine>
      <MarkedForRemoval>true</MarkedForRemoval>
      <Message>String content</Message>
      <NoteUniqueKey>String content</NoteUniqueKey>
      <NotificationType>String content</NotificationType>
    </Notification>
  </ActiveNotifications>
  <ActiveValidations>
    <Validation>
      <ElementIdentifier>String content</ElementIdentifier>
      <InvalidMessageText>String content</InvalidMessageText>
      <IsValid>true</IsValid>
      <Reasons>
        <ValidationReason>
          <FiringRuleId>String content</FiringRuleId>
          <ManagedByRuleEngine>true</ManagedByRuleEngine>
          <MarkedForRemoval>true</MarkedForRemoval>
          <MessageText>String content</MessageText>
          <OwningRuleId>String content</OwningRuleId>
        </ValidationReason>
      </Reasons>
    </Validation>
  </ActiveValidations>
  <EntityState>String content</EntityState>
  <HasRuntimeErrors>false</HasRuntimeErrors>
  <Overrides>
    <ConfiguredOverride>
      <Name>String content</Name>
      <OverrideType>DatabaseConnection</OverrideType>
      <Value>String content</Value>
    </ConfiguredOverride>
  </Overrides>
  <RequestId>1627aea5-8e0a-4371-9022-9b504344e724</RequestId>
  <RuleExecutionLog>
    <CalcsEvaluatedCount>9223372036854775807</CalcsEvaluatedCount>
    <Messages>
      <!--Valid elements of type: CollectionChangedMessage, ErrorMessage, NotificationChangeMessag
      <RuleExecutionLogMessage i:type="CollectionChangedMessage" xmlns:i="http://www.w3.org/2001/X
        <Description>String content</Description>
        <ChangeType>Added</ChangeType>
        <CollectionCount>2147483647</CollectionCount>
        <CollectionId>String content</CollectionId>
        <MemberId>String content</MemberId>
        <MemberIndex>2147483647</MemberIndex>
      </RuleExecutionLogMessage>
    </Messages>
    <PerformedIncrementalEvaluation>true</PerformedIncrementalEvaluation>
    <RulesEvaluatedCount>9223372036854775807</RulesEvaluatedCount>
    <RulesEvaluatedTrueCount>9223372036854775807</RulesEvaluatedTrueCount>
    <TotalEvaluationCycles>9223372036854775807</TotalEvaluationCycles>
    <TotalExecutionTime>P428DT10H30M12.3S</TotalExecutionTime>
    <TotalTraceFrames>2147483647</TotalTraceFrames>
  </RuleExecutionLog>
  <RuleSessionState>String content</RuleSessionState>
  <SessionId>1627aea5-8e0a-4371-9022-9b504344e724</SessionId>
</RuleEngineHttpServiceResponse>
```

Sample JSON Response

```
{
   "ActiveNotifications":[{
      "Changed":true,
      "ElementId":"String content",
      "FiredBy":["String content"],
      "IsActive":true,
      "ManagedByRuleEngine":true,
      "MarkedForRemoval":true,
      "Message":"String content",
      "NoteUniqueKey":"String content",
      "NotificationType":"String content"
   }],
   "ActiveValidations":[{
      "ElementIdentifier":"String content",
      "InvalidMessageText":"String content",
      "IsValid":true,
      "Reasons":[{
         "FiringRuleId":"String content",
         "ManagedByRuleEngine":true,
         "MarkedForRemoval":true,
         "MessageText":"String content",
         "OwningRuleId":"String content"
      }]
   }],
   "EntityState":"String content",
   "HasRuntimeErrors":false,
   "Overrides":[{
      "Name":"String content",
      "OverrideType":0,
      "Value":"String content"
   }],
   "RequestId":"1627aea5-8e0a-4371-9022-9b504344e724",
   "RuleExecutionLog":{
      "CalcsEvaluatedCount":9223372036854775807,
      "Messages":[{
         "Description":"String content",
         "ChangeType":0,
         "CollectionCount":2147483647,
         "CollectionId":"String content",
         "MemberId":"String content",
         "MemberIndex":2147483647
      }],
      "PerformedIncrementalEvaluation":true,
      "RulesEvaluatedCount":9223372036854775807,
      "RulesEvaluatedTrueCount":9223372036854775807,
      "TotalEvaluationCycles":9223372036854775807,
      "TotalExecutionTime":"P428DT10H30M12.3S",
      "TotalTraceFrames":2147483647
   },
   "RuleSessionState":"String content",
   "SessionId":"1627aea5-8e0a-4371-9022-9b504344e724"
}
```

4.7.2.1.2 Execute Decision

URL:	e.g. http://servername/InRuleRuleEngineService/HttpService.svc/ExecuteDecision
HTTP Method:	POST

See below for examples of requests / responses in both XML and JSON formats. The variations shown are:

- File-based Rule Application (deployed on irServer)
- Catalog Rule Application
  - ○ Rule Application specified by either Name or Guid
  - ○ Revision specified as Latest, Label or Revision
  - ○ Authentication using specified Username / Password, Single Sign-On or using irServer config-specified credentials
- Optional RequestId specified
- Optional CacheTimeout specified
- Optional ConnTimeout specified
- Optional Overrides specified
- Optional Output Options specified (overriding defaults)

Note: In all XML request examples below, elements must be ordered exactly as specified.

Sample XML Request: Filesystem Rule Application

```
<ExecuteDecisionRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RequestId>A1E9BB89-7B46-4F62-B9AA-62FB03D73F03</RequestId>
  <RuleApp>
    <FileName>RuleApp1.ruleappx</FileName>
  </RuleApp>
  <DecisionName>CalculateArea</DecisionName>
  <InputState>{ "height": 10, "width": 2 }</InputState>
</ExecuteDecisionRequest>
```

Sample XML Request: Catalog Rule Application: Latest Revision by Name

```
<ExecuteDecisionRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RequestId>A1E9BB89-7B46-4F62-B9AA-62FB03D73F03</RequestId>
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
      <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <DecisionName>CalculateArea</DecisionName>
  <InputState>{ "height": 10, "width": 2 }</InputState>
</ExecuteDecisionRequest>
```

Sample XML Request: Catalog Rule Application: Latest Revision by Guid

```
<ExecuteDecisionRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RequestId>A1E9BB89-7B46-4F62-B9AA-62FB03D73F03</RequestId>
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
      <Guid>5f11d845-bda1-4f79-ba67-e8cd12d065f7</Guid>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <DecisionName>CalculateArea</DecisionName>
  <InputState>{ "height": 10, "width": 2 }</InputState>
</ExecuteDecisionRequest>
```

Sample XML Request: Catalog Rule Application: Labeled Revision by Name

```
<ExecuteDecisionRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RequestId>A1E9BB89-7B46-4F62-B9AA-62FB03D73F03</RequestId>
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
      <Label>Label1</Label>
      <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <DecisionName>CalculateArea</DecisionName>
  <InputState>{ "height": 10, "width": 2 }</InputState>
</ExecuteDecisionRequest>
```

Sample XML Request: Catalog Rule Application: Specific Revision by Name

```
<ExecuteDecisionRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RequestId>A1E9BB89-7B46-4F62-B9AA-62FB03D73F03</RequestId>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
       <Revision>3</Revision>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <DecisionName>CalculateArea</DecisionName>
  <InputState>{ "height": 10, "width": 2 }</InputState>
</ExecuteDecisionRequest>
```

Sample XML Request: Catalog Rule Application: SSO Authentication (using service host account credential)

```
<ExecuteDecisionRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RequestId>A1E9BB89-7B46-4F62-B9AA-62FB03D73F03</RequestId>
  <RuleApp>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UseIntegratedSecurity>true</UseIntegratedSecurity>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <DecisionName>CalculateArea</DecisionName>
  <InputState>{ "height": 10, "width": 2 }</InputState>
</ExecuteDecisionRequest>
```

Sample XML Request: Catalog Rule Application: irServer config-specified credentials Authentication

```
<ExecuteDecisionRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RequestId>A1E9BB89-7B46-4F62-B9AA-62FB03D73F03</RequestId>
  <RuleApp>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
  </RuleApp>
  <DecisionName>CalculateArea</DecisionName>
  <InputState>{ "height": 10, "width": 2 }</InputState>
</ExecuteDecisionRequest>
```

Sample XML Request: Catalog Rule Application: RequestId, CacheTimeout, ConnTimeout, Overrides, Output Options specified

```
<ExecuteDecisionRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RequestId>A1E9BB89-7B46-4F62-B9AA-62FB03D73F03</RequestId>
  <RuleApp>
    <CacheTimeout>100</CacheTimeout>
    <ConnTimeout>120</ConnTimeout>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <!-- see Overrides for examples -->
      </Override>
    </Overrides>
    <RuleSessionOverrides>
      <ExecutionTimeout>PT40S</ExecutionTimeout>
      <MaxCycleCount>200000</MaxCycleCount>
      <Now>1999-05-31T11:20:00</Now>
    </RuleSessionOverrides>
  </RuleEngineServiceOptions>
  <RuleEngineServiceOutputTypes>
    <ActiveNotifications>true</ActiveNotifications>
    <ActiveValidations>true</ActiveValidations>
    <EntityState>true</EntityState>
    <Overrides>true</Overrides>
    <RuleExecutionLog>true</RuleExecutionLog>
  </RuleEngineServiceOutputTypes>
  <DecisionName>CalculateArea</DecisionName>
  <InputState>{ "height": 10, "width": 2 }</InputState>
</ExecuteDecisionRequest>
```

Sample JSON Request: Filesystem Rule Application

```
{
   "RequestId": "A1E9BB89-7B46-4F62-B9AA-62FB03D73F03",
   "RuleApp": {
      "FileName": "RuleApp1.ruleappx"
   },
   "DecisionName": "CalculateArea",
   "InputState": "{\"height\":10,\"width\":2}"
}
```

Sample JSON Request: Catalog Rule Application: Latest Revision by Name

```
{
   "RequestId": "A1E9BB89-7B46-4F62-B9AA-62FB03D73F03",
   "RuleApp": {
      "RepositoryRuleAppRevisionSpec": {
         "RuleApplicationName": "RuleApp1"
      },
      "RepositoryServiceUri": "http://localhost/InRuleCatalogService/service.svc",
      "UserName": "CatUsername",
      "Password": "CatPassword"
   },
   "DecisionName": "CalculateArea",
   "InputState": "{\"height\":10,\"width\":2}"
}
```

Sample JSON Request: Catalog Rule Application: Latest Revision by Guid

```
{
   "RequestId": "A1E9BB89-7B46-4F62-B9AA-62FB03D73F03",
   "RuleApp": {
      "RepositoryRuleAppRevisionSpec": {
         "Guid": "5f11d845-bda1-4f79-ba67-e8cd12d065f7"
      },
      "RepositoryServiceUri": "http://localhost/InRuleCatalogService/service.svc",
      "UserName": "CatUsername",
      "Password": "CatPassword"
   },
   "DecisionName": "CalculateArea",
   "InputState": "{\"height\":10,\"width\":2}"
}
```

Sample JSON Request: Catalog Rule Application: Labeled Revision by Name

```
{
   "RequestId": "A1E9BB89-7B46-4F62-B9AA-62FB03D73F03",
   "RuleApp": {
      "RepositoryRuleAppRevisionSpec": {
         "Label": "Label1",
         "RuleApplicationName": "RuleApp1"
      },
      "RepositoryServiceUri": "http://localhost/InRuleCatalogService/service.svc",
      "UserName": "CatUsername",
      "Password": "CatPassword"
   },
   "DecisionName": "CalculateArea",
   "InputState": "{\"height\":10,\"width\":2}"
}
```

Sample JSON Request: Catalog Rule Application: Specific Revision by Name

```
{
   "RequestId": "A1E9BB89-7B46-4F62-B9AA-62FB03D73F03",
   "RuleApp": {
      "RepositoryRuleAppRevisionSpec": {
         "Revision": 3,
         "RuleApplicationName": "RuleApp1"
      },
      "RepositoryServiceUri": "http://localhost/InRuleCatalogService/service.svc",
      "UserName": "CatUsername",
      "Password": "CatPassword"
   },
   "DecisionName": "CalculateArea",
   "InputState": "{\"height\":10,\"width\":2}"
}
```

Sample JSON Request: Catalog Rule Application: SSO Authentication (using service host account credential)

```
{
   "RequestId": "A1E9BB89-7B46-4F62-B9AA-62FB03D73F03",
   "RuleApp": {
      "RepositoryRuleAppRevisionSpec": {
         "RuleApplicationName": "RuleApp1"
      },
      "RepositoryServiceUri": "http://localhost/InRuleCatalogService/service.svc",
      "UseIntegratedSecurity": true,
      "UserName": "CatUsername"
   },
   "DecisionName": "CalculateArea",
   "InputState": "{\"height\":10,\"width\":2}"
}
```

Sample JSON Request: Catalog Rule Application: irServer config-specified credentials Authentication

```
{
   "RequestId": "A1E9BB89-7B46-4F62-B9AA-62FB03D73F03",
   "RuleApp": {
      "RepositoryRuleAppRevisionSpec": {
         "RuleApplicationName": "RuleApp1"
      },
      "RepositoryServiceUri": "http://localhost/InRuleCatalogService/service.svc"
   },
   "DecisionName": "CalculateArea",
   "InputState": "{\"height\":10,\"width\":2}"
}
```

Sample JSON Request: Catalog Rule Application: RequestId, CacheTimeout, ConnTimeout, Overrides, Output Options specified

```
{
   "RequestId": "A1E9BB89-7B46-4F62-B9AA-62FB03D73F03",
   "RuleApp": {
      "CacheTimeout": 120,
      "ConnTimeout": 100,
      "RepositoryRuleAppRevisionSpec": {
         "Revision": 3,
         "RuleApplicationName": "RuleApp1"
      },
      "RepositoryServiceUri": "http://localhost/InRuleCatalogService/service.svc",
      "UserName": "CatUsername",
      "UseIntegratedSecurity":true,
   },
   "RuleEngineServiceOptions": {
      "Overrides": [
         // see Overrides for example.
      ],
      "RuleSessionOverrides": {
         "ExecutionTimeout": "PT40S",
         "MaxCycleCount": 200000,
         "Now": "/Date(928167600000-0500)/",
      },
   },
   "RuleEngineServiceOutputTypes": {
      "ActiveNotifications": true,
      "ActiveValidations": true,
      "EntityState": true,
      "Overrides": true,
      "RuleExecutionLog": true
   },
   "DecisionName": "CalculateArea",
   "InputState": "{\"height\":10,\"width\":2}"
}
```

Sample XML Response

```
<ExecuteDecisionResponse xmlns="http://www.inrule.com/XmlSchema/Schema">
  <ActiveNotifications>
    <Notification>
      <Changed>true</Changed>
      <ElementId>String content</ElementId>
      <FiredBy>
        <string xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">String content</
        <string xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">String content</
      </FiredBy>
      <IsActive>true</IsActive>
      <ManagedByRuleEngine>true</ManagedByRuleEngine>
      <MarkedForRemoval>true</MarkedForRemoval>
      <Message>String content</Message>
      <NoteUniqueKey>String content</NoteUniqueKey>
      <NotificationType>String content</NotificationType>
    </Notification>
  </ActiveNotifications>
  <ActiveValidations>
    <Validation>
      <ElementIdentifier>String content</ElementIdentifier>
      <InvalidMessageText>String content</InvalidMessageText>
      <IsValid>true</IsValid>
      <Reasons>
        <ValidationReason>
          <FiringRuleId>String content</FiringRuleId>
          <ManagedByRuleEngine>true</ManagedByRuleEngine>
          <MarkedForRemoval>true</MarkedForRemoval>
          <MessageText>String content</MessageText>
          <OwningRuleId>String content</OwningRuleId>
        </ValidationReason>
      </Reasons>
    </Validation>
  </ActiveValidations>
  <HasRuntimeErrors>false</HasRuntimeErrors>
  <OutputState>{"area":20}</OutputState>
  <RequestId>a1e9bb89-7b46-4f62-b9aa-62fb03d73f03</RequestId>
  <RuleExecutionLog>
    <CalcsEvaluatedCount>9223372036854775807</CalcsEvaluatedCount>
    <Messages>
      <!--Valid elements of type: CollectionChangedMessage, ErrorMessage, NotificationChangeMessag
    </Messages>
    <PerformedIncrementalEvaluation>true</PerformedIncrementalEvaluation>
    <RulesEvaluatedCount>9223372036854775807</RulesEvaluatedCount>
    <RulesEvaluatedTrueCount>9223372036854775807</RulesEvaluatedTrueCount>
    <TotalEvaluationCycles>9223372036854775807</TotalEvaluationCycles>
    <TotalExecutionTime>P428DT10H30M12.3S</TotalExecutionTime>
    <TotalTraceFrames>2147483647</TotalTraceFrames>
  </RuleExecutionLog>
  <SessionId>1627aea5-8e0a-4371-9022-9b504344e724</SessionId>
</ExecuteDecisionResponse>
```

Sample JSON Response

```
{
   "ActiveNotifications": [{
      "Changed": true,
      "ElementId": "String content",
      "FiredBy": ["String content"],
      "IsActive": true,
      "ManagedByRuleEngine": true,
      "MarkedForRemoval": true,
      "Message": "String content",
      "NoteUniqueKey": "String content",
      "NotificationType": "String content"
   }],
   "ActiveValidations":[{
      "ElementIdentifier": "String content",
      "InvalidMessageText": "String content",
      "IsValid": true,
      "Reasons": [{
         "FiringRuleId": "String content",
         "ManagedByRuleEngine": true,
         "MarkedForRemoval": true,
         "MessageText": "String content",
         "OwningRuleId": "String content"
      }]
   }],
   "HasRuntimeErrors": false,
   "OutputState": "{\"area\":20}",
   "RequestId": "a1e9bb89-7b46-4f62-b9aa-62fb03d73f03",
   "RuleExecutionLog": {
      "CalcsEvaluatedCount": 9223372036854775807,
      "Messages": [{
         "Description": "String content",
         "ChangeType": 0,
         "CollectionCount": 2147483647,
         "CollectionId": "String content",
         "MemberId": "String content",
         "MemberIndex": 2147483647
      }],
      "PerformedIncrementalEvaluation": true,
      "RulesEvaluatedCount": 9223372036854775807,
      "RulesEvaluatedTrueCount": 9223372036854775807,
      "TotalEvaluationCycles": 9223372036854775807,
      "TotalExecutionTime": "P428DT10H30M12.3S",
      "TotalTraceFrames": 2147483647
   },
   "SessionId":"1627aea5-8e0a-4371-9022-9b504344e724"
}
```

4.7.2.1.3 Execute Independent Rule Set

URL:	e.g. http://servername/InRuleRuleEngineService/HttpService.svc/ExecuteIndependentRuleSet
HTTP Method:	POST

See below for examples of requests / responses in both XML and JSON formats. The variations shown are:

- File-based Rule Application (deployed on irServer)

- Catalog Rule Application
  - Rule Application specified by either Name or Guid
  - Revision specified as Latest, Label or Revision
  - Authentication using specified Username / Password, Single Sign-On or using irServer configuration-specified credentials (see InRule Runtime Service Config File Settings)
- Optional RequestId specified
- Optional CacheTimeout specified
- Optional ConnTimeout specified
- Optional Overrides specified
- Optional Output Options specified (overriding defaults)

Note: In all XML request examples below, elements must be ordered exactly as specified.

Sample XML Request: Filesystem Rule Application

```
<ExecuteIndependentRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <FileName>Ruleapp1.ruleapp</FileName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <RuleSetName>Ruleset1</RuleSetName>
  <EntityStateFieldName>Entity1Param</EntityStateFieldName>
</ExecuteIndependentRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: Latest Revision by Name

```
<ExecuteIndependentRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
      <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <RuleSetName>Ruleset1</RuleSetName>
  <EntityStateFieldName>Entity1Param</EntityStateFieldName>
</ExecuteIndependentRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: Latest Revision by Guid

```xml
<ExecuteIndependentRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
      <Guid>5f11d845-bda1-4f79-ba67-e8cd12d065f7</Guid>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <RuleSetName>Ruleset1</RuleSetName>
  <EntityStateFieldName>Entity1Param</EntityStateFieldName>
</ExecuteIndependentRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: Labeled Revision by Name

```xml
<ExecuteIndependentRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
      <Label>Label1</Label>
      <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <RuleSetName>Ruleset1</RuleSetName>
  <EntityStateFieldName>Entity1Param</EntityStateFieldName>
</ExecuteIndependentRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: Specific Revision by Name

```xml
<ExecuteIndependentRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
       <Revision>3</Revision>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <RuleSetName>Ruleset1</RuleSetName>
  <EntityStateFieldName>Entity1Param</EntityStateFieldName>
</ExecuteIndependentRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: SSO Authentication (using service host account credential)

```
<ExecuteIndependentRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UseIntegratedSecurity>true</UseIntegratedSecurity>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <RuleSetName>Ruleset1</RuleSetName>
  <EntityStateFieldName>Entity1Param</EntityStateFieldName>
</ExecuteIndependentRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: irServer config-specified credentials Authentication

```
<ExecuteIndependentRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <RuleSetName>Ruleset1</RuleSetName>
  <EntityStateFieldName>Entity1Param</EntityStateFieldName>
</ExecuteIndependentRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: RequestId, CacheTimeout, ConnTimeout, Overrides, Output Options specified

```
<ExecuteIndependentRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RequestId>1627aea5-8e0a-4371-9022-9b504344e724</RequestId>
  <RuleApp>
    <CacheTimeout>100</CacheTimeout>
    <ConnTimeout>120</ConnTimeout>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <!-- see Overrides for examples -->
      </Override>
    </Overrides>
    <RuleSessionOverrides>
      <ExecutionTimeout>PT40S</ExecutionTimeout>
      <MaxCycleCount>200000</MaxCycleCount>
      <Now>1999-05-31T11:20:00</Now>
    </RuleSessionOverrides>
  </RuleEngineServiceOptions>
  <RuleEngineServiceOutputTypes>
    <ActiveNotifications>true</ActiveNotifications>
    <ActiveValidations>true</ActiveValidations>
    <EntityState>true</EntityState>
    <Overrides>true</Overrides>
    <RuleExecutionLog>true</RuleExecutionLog>
  </RuleEngineServiceOutputTypes>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <RuleSetName>Ruleset1</RuleSetName>
  <EntityStateFieldName>Entity1Param</EntityStateFieldName>
</ExecuteIndependentRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: RequestId, CacheTimeout, ConnTimeout, Overrides, Output Options, Rule Set Parameters specified

```
<ExecuteIndependentRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RequestId>1627aea5-8e0a-4371-9022-9b504344e724</RequestId>
  <RuleApp>
    <CacheTimeout>100</CacheTimeout>
    <ConnTimeout>120</ConnTimeout>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <!-- see Overrides for examples -->
      </Override>
    </Overrides>
    <RuleSessionOverrides>
      <ExecutionTimeout>PT40S</ExecutionTimeout>
      <MaxCycleCount>200000</MaxCycleCount>
      <Now>1999-05-31T11:20:00</Now>
    </RuleSessionOverrides>
  </RuleEngineServiceOptions>
  <RuleEngineServiceOutputTypes>
    <ActiveNotifications>true</ActiveNotifications>
    <ActiveValidations>true</ActiveValidations>
    <EntityState>true</EntityState>
    <Overrides>true</Overrides>
    <RuleExecutionLog>true</RuleExecutionLog>
  </RuleEngineServiceOutputTypes>
  <Parameters>
    <Parameter>
      <Name>Parameter1</Name>
      <Value>Value1</Value>
    </Parameter>
    <Parameter>
      <Name>Parameter2</Name>
      <Value>Value2</Value>
    </Parameter>
  </Parameters>
  <RuleSetName>Ruleset1</RuleSetName>
</ExecuteIndependentRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: RequestId, CacheTimeout, ConnTimeout, Overrides, Output Options, Entity Rule Set Parameters specified

```
<ExecuteIndependentRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RequestId>1627aea5-8e0a-4371-9022-9b504344e724</RequestId>
  <RuleApp>
    <CacheTimeout>100</CacheTimeout>
    <ConnTimeout>120</ConnTimeout>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <!-- see Overrides for examples -->
      </Override>
    </Overrides>
    <RuleSessionOverrides>
      <ExecutionTimeout>PT40S</ExecutionTimeout>
      <MaxCycleCount>200000</MaxCycleCount>
      <Now>1999-05-31T11:20:00</Now>
    </RuleSessionOverrides>
  </RuleEngineServiceOptions>
  <RuleEngineServiceOutputTypes>
    <ActiveNotifications>true</ActiveNotifications>
    <ActiveValidations>true</ActiveValidations>
    <EntityState>true</EntityState>
    <Overrides>true</Overrides>
    <RuleExecutionLog>true</RuleExecutionLog>
  </RuleEngineServiceOutputTypes>
  <Parameters>
    <Parameter>
      <Name>Parameter1</Name>
      <Value>Value1</Value>
    </Parameter>
    <Parameter>
      <Name>EntityParameter2</Name>
      <Value>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity2 &gt;&lt;Field1&gt;1&lt;/Field
    </Parameter>
  </Parameters>
  <RuleSetName>Ruleset1</RuleSetName>
</ExecuteIndependentRuleSetRequest>
```

Sample JSON Request: Filesystem Rule Application

```
{
   "RuleApp":{
      "FileName":"Ruleapp1.ruleapp"
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "RuleSetName":"Ruleset1",
   "EntityStateFieldName":"Entity1Param"
}
```

Sample JSON Request: Catalog Rule Application: Latest Revision by Name

```
{
   "RuleApp":{
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UserName":"CatUsername"
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "RuleSetName":"Ruleset1",
   "EntityStateFieldName":"Entity1Param"
}
```

Sample JSON Request: Catalog Rule Application: Latest Revision by Guid

```
{
   "RuleApp":{
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "Guid":"5f11d845-bda1-4f79-ba67-e8cd12d065f7"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UserName":"CatUsername"
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "RuleSetName":"Ruleset1",
   "EntityStateFieldName":"Entity1Param"
}
```

Sample JSON Request: Catalog Rule Application: Labeled Revision by Name

```
{
   "RuleApp":{
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "Label":"Label1",
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UserName":"CatUsername"
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "RuleSetName":"Ruleset1",
   "EntityStateFieldName":"Entity1Param"
}
```

Sample JSON Request: Catalog Rule Application: Specific Revision by Name

```
{
   "RuleApp":{
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "Revision":3,
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UserName":"CatUsername"
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "RuleSetName":"Ruleset1",
   "EntityStateFieldName":"Entity1Param"
}
```

Sample JSON Request: Catalog Rule Application: SSO Authentication (using service host account credential)

```
{
   "RuleApp":{
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UseIntegratedSecurity":true
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "RuleSetName":"Ruleset1",
   "EntityStateFieldName":"Entity1Param"
}
```

Sample JSON Request: Catalog Rule Application: irServer config-specified credentials Authentication

```
{
   "RuleApp":{
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc"
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "RuleSetName":"Ruleset1",
   "EntityStateFieldName":"Entity1Param"
}
```

Sample JSON Request: Catalog Rule Application: RequestId, CacheTimeout, ConnTimeout, Overrides, Output Options specified

```
{
   "RequestId":"1627aea5-8e0a-4371-9022-9b504344e724",
   "RuleApp":{
      "CacheTimeout":"120",
      "ConnTimeout":"100",
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UseIntegratedSecurity":true,
      "UserName":"CatPassword"
   },
   "RuleEngineServiceOptions":{
      "Overrides":[
         // see Overrides for example.
      ],
      "RuleSessionOverrides":{
         "ExecutionTimeout":"PT40S",
         "MaxCycleCount":200000,
         "Now":"\/Date(928167600000-0500)\/",
      },
   },
   "RuleEngineServiceOutputTypes":{
      "ActiveNotifications":true,
      "ActiveValidations":true,
      "EntityState":true,
      "Overrides":true,
      "RuleExecutionLog":true
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "RuleSetName":"Ruleset1",
   "EntityStateFieldName":"Entity1Param"
}
```

Sample JSON Request: Catalog Rule Application: RequestId, CacheTimeout, ConnTimeout, Overrides, Output Options, Rule Set Parameters specified

```
{
   "RequestId":"1627aea5-8e0a-4371-9022-9b504344e724",
   "RuleApp":{
      "CacheTimeout":"120",
      "ConnTimeout":"100",
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UseIntegratedSecurity":true,
      "UserName":"CatPassword"
   },
   "RuleEngineServiceOptions":{
      "Overrides":[
         // see Overrides for example.
      ],
      "RuleSessionOverrides":{
         "ExecutionTimeout":"PT40S",
         "MaxCycleCount":200000,
         "Now":"\/Date(928167600000-0500)\/",
      },
   },
   "RuleEngineServiceOutputTypes":{
      "ActiveNotifications":true,
      "ActiveValidations":true,
      "EntityState":true,
      "Overrides":true,
      "RuleExecutionLog":true
   },
   "Parameters": [
      {"Name":"Parameter1","Value":"Value1"},
      {"Name":"Parameter2","Value":"Value2"}
    ],
   "RuleSetName":"Ruleset1"
}
```

Sample JSON Request: Catalog Rule Application: RequestId, CacheTimeout, ConnTimeout, Overrides, Output Options, Entity Rule Set Parameters specified

```
{
   "RequestId":"1627aea5-8e0a-4371-9022-9b504344e724",
   "RuleApp":{
      "CacheTimeout":"120",
      "ConnTimeout":"100",
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UseIntegratedSecurity":true,
      "UserName":"CatPassword"
   },
   "RuleEngineServiceOptions":{
      "Overrides":[
         // see Overrides for example.
      ],
      "RuleSessionOverrides":{
         "ExecutionTimeout":"PT40S",
         "MaxCycleCount":200000,
         "Now":"\/Date(928167600000-0500)\/",
      },
   },
   "RuleEngineServiceOutputTypes":{
      "ActiveNotifications":true,
      "ActiveValidations":true,
      "EntityState":true,
      "Overrides":true,
      "RuleExecutionLog":true
   },
   "Parameters": [
      {"Name":"Parameter1","Value":"Value1"},
      {"Name":"EntityParameter2","Value":"{\"Field1\":\"1\"}"}
    ],
   "RuleSetName":"Ruleset1"
}
```

Sample XML Response

```
<RuleEngineHttpServiceResponse xmlns="http://www.inrule.com/XmlSchema/Schema">
  <ActiveNotifications>
    <Notification>
      <Changed>true</Changed>
      <ElementId>String content</ElementId>
      <FiredBy>
        <string xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">String content</
        <string xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">String content</
      </FiredBy>
      <IsActive>true</IsActive>
      <ManagedByRuleEngine>true</ManagedByRuleEngine>
      <MarkedForRemoval>true</MarkedForRemoval>
      <Message>String content</Message>
      <NoteUniqueKey>String content</NoteUniqueKey>
      <NotificationType>String content</NotificationType>
    </Notification>
  </ActiveNotifications>
  <ActiveValidations>
    <Validation>
      <ElementIdentifier>String content</ElementIdentifier>
      <InvalidMessageText>String content</InvalidMessageText>
      <IsValid>true</IsValid>
      <Reasons>
        <ValidationReason>
          <FiringRuleId>String content</FiringRuleId>
          <ManagedByRuleEngine>true</ManagedByRuleEngine>
          <MarkedForRemoval>true</MarkedForRemoval>
          <MessageText>String content</MessageText>
          <OwningRuleId>String content</OwningRuleId>
        </ValidationReason>
      </Reasons>
    </Validation>
  </ActiveValidations>
  <EntityState>String content</EntityState>
  <HasRuntimeErrors>false</HasRuntimeErrors>
  <Overrides>
    <ConfiguredOverride>
      <Name>String content</Name>
      <OverrideType>DatabaseConnection</OverrideType>
      <Value>String content</Value>
    </ConfiguredOverride>
  </Overrides>
  <RequestId>1627aea5-8e0a-4371-9022-9b504344e724</RequestId>
  <RuleExecutionLog>
    <CalcsEvaluatedCount>9223372036854775807</CalcsEvaluatedCount>
    <Messages>
      <!--Valid elements of type: CollectionChangedMessage, ErrorMessage, NotificationChangeMessag
      <RuleExecutionLogMessage i:type="CollectionChangedMessage" xmlns:i="http://www.w3.org/2001/X
        <Description>String content</Description>
        <ChangeType>Added</ChangeType>
        <CollectionCount>2147483647</CollectionCount>
        <CollectionId>String content</CollectionId>
        <MemberId>String content</MemberId>
        <MemberIndex>2147483647</MemberIndex>
      </RuleExecutionLogMessage>
    </Messages>
    <PerformedIncrementalEvaluation>true</PerformedIncrementalEvaluation>
    <RulesEvaluatedCount>9223372036854775807</RulesEvaluatedCount>
    <RulesEvaluatedTrueCount>9223372036854775807</RulesEvaluatedTrueCount>
    <TotalEvaluationCycles>9223372036854775807</TotalEvaluationCycles>
    <TotalExecutionTime>P428DT10H30M12.3S</TotalExecutionTime>
    <TotalTraceFrames>2147483647</TotalTraceFrames>
  </RuleExecutionLog>
  <RuleSessionState>String content</RuleSessionState>
  <SessionId>1627aea5-8e0a-4371-9022-9b504344e724</SessionId>
</RuleEngineHttpServiceResponse>
```

Sample JSON Response

```
{
   "ActiveNotifications":[{
      "Changed":true,
      "ElementId":"String content",
      "FiredBy":["String content"],
      "IsActive":true,
      "ManagedByRuleEngine":true,
      "MarkedForRemoval":true,
      "Message":"String content",
      "NoteUniqueKey":"String content",
      "NotificationType":"String content"
   }],
   "ActiveValidations":[{
      "ElementIdentifier":"String content",
      "InvalidMessageText":"String content",
      "IsValid":true,
      "Reasons":[{
         "FiringRuleId":"String content",
         "ManagedByRuleEngine":true,
         "MarkedForRemoval":true,
         "MessageText":"String content",
         "OwningRuleId":"String content"
      }]
   }],
   "EntityState":"String content",
   "HasRuntimeErrors":false,
   "Overrides":[{
      "Name":"String content",
      "OverrideType":0,
      "Value":"String content"
   }],
   "RequestId":"1627aea5-8e0a-4371-9022-9b504344e724",
   "RuleExecutionLog":{
      "CalcsEvaluatedCount":9223372036854775807,
      "Messages":[{
         "Description":"String content",
         "ChangeType":0,
         "CollectionCount":2147483647,
         "CollectionId":"String content",
         "MemberId":"String content",
         "MemberIndex":2147483647
      }],
      "PerformedIncrementalEvaluation":true,
      "RulesEvaluatedCount":9223372036854775807,
      "RulesEvaluatedTrueCount":9223372036854775807,
      "TotalEvaluationCycles":9223372036854775807,
      "TotalExecutionTime":"P428DT10H30M12.3S",
      "TotalTraceFrames":2147483647
   },
   "RuleSessionState":"String content",
   "SessionId":"1627aea5-8e0a-4371-9022-9b504344e724"
}
```

4.7.2.1.4 Execute Rule Set

URL:	e.g. http://servername/InRuleRuleEngineService/HttpService.svc/ExecuteRuleSet
HTTP Method:	POST

See below for examples of requests / responses in both XML and JSON formats. The variations shown are:

- File-based Rule Application (deployed on irServer)
- Catalog Rule Application
  - Rule Application specified by either Name or Guid
  - Revision specified as Latest, Label or Revision
  - Authentication using specified Username / Password, Single Sign-On or using irServer config-specified credentials
- Optional RequestId specified
- Optional CacheTimeout specified
- Optional ConnTimeout specified
- Optional Overrides specified
- Optional Output Options specified (overriding defaults)
- Optional Rule Set Parameters specified

Note: In all XML request examples below, elements must be ordered exactly as specified.

Sample XML Request: Filesystem Rule Application

```
<ExecuteRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <FileName>Ruleapp1.ruleapp</FileName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
  <RuleSetName>Ruleset1</RuleSetName>
</ExecuteRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: Latest Revision by Name

```
<ExecuteRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
      <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
  <RuleSetName>Ruleset1</RuleSetName>
</ExecuteRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: Latest Revision by Guid

```
<ExecuteRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
      <Guid>5f11d845-bda1-4f79-ba67-e8cd12d065f7</Guid>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
  <RuleSetName>Ruleset1</RuleSetName>
</ExecuteRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: Labeled Revision by Name

```
<ExecuteRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
      <Label>Label1</Label>
      <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
  <RuleSetName>Ruleset1</RuleSetName>
</ExecuteRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: Specific Revision by Name

```
<ExecuteRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
       <Revision>3</Revision>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
  <RuleSetName>Ruleset1</RuleSetName>
</ExecuteRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: SSO Authentication (using service host account credential)

```
<ExecuteRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UseIntegratedSecurity>true</UseIntegratedSecurity>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
  <RuleSetName>Ruleset1</RuleSetName>
</ExecuteRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: irServer config-specified credentials Authentication

```
<ExecuteRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RuleApp>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
  </RuleApp>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
  <RuleSetName>Ruleset1</RuleSetName>
</ExecuteRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: RequestId, CacheTimeout, ConnTimeout, Overrides, Output Options, Rule Set Parameters specified

```
<ExecuteRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RequestId>1627aea5-8e0a-4371-9022-9b504344e724</RequestId>
  <RuleApp>
    <CacheTimeout>100</CacheTimeout>
    <ConnTimeout>120</ConnTimeout>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <!-- see Overrides for examples -->
      </Override>
    </Overrides>
    <RuleSessionOverrides>
      <ExecutionTimeout>PT40S</ExecutionTimeout>
      <MaxCycleCount>200000</MaxCycleCount>
      <Now>1999-05-31T11:20:00</Now>
    </RuleSessionOverrides>
  </RuleEngineServiceOptions>
  <RuleEngineServiceOutputTypes>
    <ActiveNotifications>true</ActiveNotifications>
    <ActiveValidations>true</ActiveValidations>
    <EntityState>true</EntityState>
    <Overrides>true</Overrides>
    <RuleExecutionLog>true</RuleExecutionLog>
  </RuleEngineServiceOutputTypes>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
  <Parameters>
    <Parameter>
      <Name>Parameter1</Name>
      <Value>Value1</Value>
    </Parameter>
    <Parameter>
      <Name>Parameter2</Name>
      <Value>Value2</Value>
    </Parameter>
  </Parameters>
  <RuleSetName>Ruleset1</RuleSetName>
</ExecuteRuleSetRequest>
```

Sample XML Request: Catalog Rule Application: RequestId, CacheTimeout, ConnTimeout, Overrides, Output Options, Entity Rule Set Parameters specified

```
<ExecuteRuleSetRequest xmlns="http://www.inrule.com/XmlSchema/Schema">
  <RequestId>1627aea5-8e0a-4371-9022-9b504344e724</RequestId>
  <RuleApp>
    <CacheTimeout>100</CacheTimeout>
    <ConnTimeout>120</ConnTimeout>
    <Password>CatPassword</Password>
    <RepositoryRuleAppRevisionSpec>
       <RuleApplicationName>RuleApp1</RuleApplicationName>
    </RepositoryRuleAppRevisionSpec>
    <RepositoryServiceUri>http://localhost/InRuleCatalogService/service.svc</RepositoryServiceUri>
    <UserName>CatUsername</UserName>
  </RuleApp>
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <!-- see Overrides for examples -->
      </Override>
    </Overrides>
    <RuleSessionOverrides>
      <ExecutionTimeout>PT40S</ExecutionTimeout>
      <MaxCycleCount>200000</MaxCycleCount>
      <Now>1999-05-31T11:20:00</Now>
    </RuleSessionOverrides>
  </RuleEngineServiceOptions>
  <RuleEngineServiceOutputTypes>
    <ActiveNotifications>true</ActiveNotifications>
    <ActiveValidations>true</ActiveValidations>
    <EntityState>true</EntityState>
    <Overrides>true</Overrides>
    <RuleExecutionLog>true</RuleExecutionLog>
  </RuleEngineServiceOutputTypes>
  <EntityState>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity1 &gt;&lt;Field1&gt;1&lt;/Fie
  <EntityName>Entity1</EntityName>
  <Parameters>
    <Parameter>
      <Name>Parameter1</Name>
      <Value>Value1</Value>
    </Parameter>
    <Parameter>
      <Name>EntityParameter2</Name>
      <Value>&lt;?xml version='1.0' encoding='utf-8'?&gt;&lt;Entity2 &gt;&lt;Field1&gt;1&lt;/Fiel
    </Parameter>
  </Parameters>
  <RuleSetName>Ruleset1</RuleSetName>
</ExecuteRuleSetRequest>
```

Sample JSON Request: Filesystem Rule Application

```
{
   "RuleApp":{
      "FileName":"Ruleapp1.ruleapp"
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "EntityName":"Entity1",
   "RuleSetName":"Ruleset1"
}
```

Sample JSON Request: Catalog Rule Application: Latest Revision by Name

```
{
   "RuleApp":{
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UserName":"CatUsername"
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "EntityName":"Entity1",
   "RuleSetName":"Ruleset1"
}
```

Sample JSON Request: Catalog Rule Application: Latest Revision by Guid

```
{
   "RuleApp":{
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "Guid":"5f11d845-bda1-4f79-ba67-e8cd12d065f7"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UserName":"CatUsername"
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "EntityName":"Entity1",
   "RuleSetName":"Ruleset1"
}
```

Sample JSON Request: Catalog Rule Application: Labeled Revision by Name

```
{
   "RuleApp":{
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "Label":"Label1",
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UserName":"CatUsername"
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "EntityName":"Entity1",
   "RuleSetName":"Ruleset1"
}
```

Sample JSON Request: Catalog Rule Application: Specific Revision by Name

```
{
   "RuleApp":{
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "Revision":3,
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UserName":"CatUsername"
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "EntityName":"Entity1",
   "RuleSetName":"Ruleset1"
}
```

Sample JSON Request: Catalog Rule Application: SSO Authentication (using service host account credential)

```
{
   "RuleApp":{
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UseIntegratedSecurity":true
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "EntityName":"Entity1",
   "RuleSetName":"Ruleset1"
}
```

Sample JSON Request: Catalog Rule Application: irServer config-specified credentials Authentication

```
{
   "RuleApp":{
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc"
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "EntityName":"Entity1",
   "RuleSetName":"Ruleset1"
}
```

Sample JSON Request: Catalog Rule Application: RequestId, CacheTimeout, ConnTimeout, Overrides, Output Options, Rule Set Parameters specified

```
{
   "RequestId":"1627aea5-8e0a-4371-9022-9b504344e724",
   "RuleApp":{
      "CacheTimeout":"120",
      "ConnTimeout":"100",
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UseIntegratedSecurity":true,
      "UserName":"CatPassword"
   },
   "RuleEngineServiceOptions":{
      "Overrides":[
         // see Overrides for example.
      ],
      "RuleSessionOverrides":{
         "ExecutionTimeout":"PT40S",
         "MaxCycleCount":200000,
         "Now":"\/Date(928167600000-0500)\/",
      },
   },
   "RuleEngineServiceOutputTypes":{
      "ActiveNotifications":true,
      "ActiveValidations":true,
      "EntityState":true,
      "Overrides":true,
      "RuleExecutionLog":true
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "EntityName":"Entity1",
   "Parameters": [
      {"Name":"Parameter1","Value":"Value1"},
      {"Name":"Parameter2","Value":"Value2"}
    ],
   "RuleSetName":"Ruleset1"
}
```

Sample JSON Request: Catalog Rule Application: RequestId, CacheTimeout, ConnTimeout, Overrides, Output Options, Entity Rule Set Parameters specified

```
{
   "RequestId":"1627aea5-8e0a-4371-9022-9b504344e724",
   "RuleApp":{
      "CacheTimeout":"120",
      "ConnTimeout":"100",
      "Password":"CatPassword",
      "RepositoryRuleAppRevisionSpec":{
         "RuleApplicationName":"Ruleapp1"
      },
      "RepositoryServiceUri":"http:\/\/localhost\/InRuleCatalogService\/service.svc",
      "UseIntegratedSecurity":true,
      "UserName":"CatPassword"
   },
   "RuleEngineServiceOptions":{
      "Overrides":[
         // see Overrides for example.
      ],
      "RuleSessionOverrides":{
         "ExecutionTimeout":"PT40S",
         "MaxCycleCount":200000,
         "Now":"\/Date(928167600000-0500)\/",
      },
   },
   "RuleEngineServiceOutputTypes":{
      "ActiveNotifications":true,
      "ActiveValidations":true,
      "EntityState":true,
      "Overrides":true,
      "RuleExecutionLog":true
   },
   "EntityState":"{\"Field1\":\"1\"}",
   "EntityName":"Entity1",
   "Parameters": [
      {"Name":"Parameter1","Value":"Value1"},
      {"Name":"EntityParameter2","Value":"{\"Field1\":\"1\"}"}
    ],
   "RuleSetName":"Ruleset1"
}
```

Sample XML Response

```
<RuleEngineHttpServiceResponse xmlns="http://www.inrule.com/XmlSchema/Schema">
  <ActiveNotifications>
    <Notification>
      <Changed>true</Changed>
      <ElementId>String content</ElementId>
      <FiredBy>
        <string xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">String content</
        <string xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">String content</
      </FiredBy>
      <IsActive>true</IsActive>
      <ManagedByRuleEngine>true</ManagedByRuleEngine>
      <MarkedForRemoval>true</MarkedForRemoval>
      <Message>String content</Message>
      <NoteUniqueKey>String content</NoteUniqueKey>
      <NotificationType>String content</NotificationType>
    </Notification>
  </ActiveNotifications>
  <ActiveValidations>
    <Validation>
      <ElementIdentifier>String content</ElementIdentifier>
      <InvalidMessageText>String content</InvalidMessageText>
      <IsValid>true</IsValid>
      <Reasons>
        <ValidationReason>
          <FiringRuleId>String content</FiringRuleId>
          <ManagedByRuleEngine>true</ManagedByRuleEngine>
          <MarkedForRemoval>true</MarkedForRemoval>
          <MessageText>String content</MessageText>
          <OwningRuleId>String content</OwningRuleId>
        </ValidationReason>
      </Reasons>
    </Validation>
  </ActiveValidations>
  <EntityState>String content</EntityState>
  <HasRuntimeErrors>false</HasRuntimeErrors>
  <Overrides>
    <ConfiguredOverride>
      <Name>String content</Name>
      <OverrideType>DatabaseConnection</OverrideType>
      <Value>String content</Value>
    </ConfiguredOverride>
  </Overrides>
  <RequestId>1627aea5-8e0a-4371-9022-9b504344e724</RequestId>
  <RuleExecutionLog>
    <CalcsEvaluatedCount>9223372036854775807</CalcsEvaluatedCount>
    <Messages>
      <!--Valid elements of type: CollectionChangedMessage, ErrorMessage, NotificationChangeMessag
      <RuleExecutionLogMessage i:type="CollectionChangedMessage" xmlns:i="http://www.w3.org/2001/
        <Description>String content</Description>
        <ChangeType>Added</ChangeType>
        <CollectionCount>2147483647</CollectionCount>
        <CollectionId>String content</CollectionId>
        <MemberId>String content</MemberId>
        <MemberIndex>2147483647</MemberIndex>
      </RuleExecutionLogMessage>
    </Messages>
    <PerformedIncrementalEvaluation>true</PerformedIncrementalEvaluation>
    <RulesEvaluatedCount>9223372036854775807</RulesEvaluatedCount>
    <RulesEvaluatedTrueCount>9223372036854775807</RulesEvaluatedTrueCount>
    <TotalEvaluationCycles>9223372036854775807</TotalEvaluationCycles>
    <TotalExecutionTime>P428DT10H30M12.3S</TotalExecutionTime>
    <TotalTraceFrames>2147483647</TotalTraceFrames>
  </RuleExecutionLog>
  <RuleSessionState>String content</RuleSessionState>
  <SessionId>1627aea5-8e0a-4371-9022-9b504344e724</SessionId>
</RuleEngineHttpServiceResponse>
```

Sample JSON Response

```
{
   "ActiveNotifications":[{
      "Changed":true,
      "ElementId":"String content",
      "FiredBy":["String content"],
      "IsActive":true,
      "ManagedByRuleEngine":true,
      "MarkedForRemoval":true,
      "Message":"String content",
      "NoteUniqueKey":"String content",
      "NotificationType":"String content"
   }],
   "ActiveValidations":[{
      "ElementIdentifier":"String content",
      "InvalidMessageText":"String content",
      "IsValid":true,
      "Reasons":[{
         "FiringRuleId":"String content",
         "ManagedByRuleEngine":true,
         "MarkedForRemoval":true,
         "MessageText":"String content",
         "OwningRuleId":"String content"
      }]
   }],
   "EntityState":"String content",
   "HasRuntimeErrors":false,
   "Overrides":[{
      "Name":"String content",
      "OverrideType":0,
      "Value":"String content"
   }],
   "RequestId":"1627aea5-8e0a-4371-9022-9b504344e724",
   "RuleExecutionLog":{
      "CalcsEvaluatedCount":9223372036854775807,
      "Messages":[{
         "Description":"String content",
         "ChangeType":0,
         "CollectionCount":2147483647,
         "CollectionId":"String content",
         "MemberId":"String content",
         "MemberIndex":2147483647
      }],
      "PerformedIncrementalEvaluation":true,
      "RulesEvaluatedCount":9223372036854775807,
      "RulesEvaluatedTrueCount":9223372036854775807,
      "TotalEvaluationCycles":9223372036854775807,
      "TotalExecutionTime":"P428DT10H30M12.3S",
      "TotalTraceFrames":2147483647
   },
   "RuleSessionState":"String content",
   "SessionId":"1627aea5-8e0a-4371-9022-9b504344e724"
}
```

4.7.2.2 HTTP Request Member Definitions

Member Elements

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td><b>RequestId</b></td><td>(Guid) A GUID that is sent back with the response that can be used as a correlation identifier.
<b>RequestId</b> is optional.</td></tr>
<tr><td><b>RuleApp</b></td><td>Parent element containing information used to load the rule application from disk or irCatalog.</td></tr>
<tr><td><b>RuleEngineServiceOptions</b></td><td>Parent element containing override options for Session, Endpoints and Data Elements.
<b>RuleEngineServiceOptions</b> are optional.</td></tr>
<tr><td><b>RuleEngineServiceOutputTypes</b></td><td>Parent element defining filters for the outgoing response.
<b>RuleEngineServiceOutputTypes</b> are optional.</td></tr>
<tr><td><b>EntityState</b></td><td>(string) An encoded XML representation of an entity that rules are going to be executed against.
<b>EntityState</b> must contain an xml declaration tag:
<b><?xml version="1.0" encoding="UTF-8" ?></b>
When encoded inside of JSON, quotes need to be escaped as double quotes.
When encoded inside of XML, standard XML escaping applies:<table><tr><th>Character</th><th>Replacement</th></tr><tr><td>"</td><td>&quot;</td></tr><tr><td>'</td><td>&apos;</td></tr><tr><td><</td><td>&lt;</td></tr><tr><td>></td><td>&gt;</td></tr><tr><td>&</td><td>&amp;</td></tr></table>Used by ApplyRules, ExecuteIndependentRuleSet and ExecuteRuleSet.</td></tr>
<tr><td><b>EntityName</b></td><td>(string) Name of the XML encoded entity in <b>EntityState</b>.
Used by ApplyRules and ExecuteRuleSet.</td></tr>
<tr><td><b>RuleSetName</b></td><td>(string) Independent or Entity RuleSet name to execute.
Used by ExecuteIndependentRuleSet and ExecuteRuleSet.</td></tr>
<tr><td><b>EntityStateFieldName</b></td><td>(string) Name of the entity parameter of the Independent Rule Set which will be loaded from <b>EntityState.</b>
<b>Note: This property is deprecated after InRule version 5.0.28. Going forward use the Parameters array to pass in Rule Set Input Parameters.</b>
Used by ExecuteIndependentRuleSet.</td></tr>
<tr><td><b>Parameters</b></td><td>(Parameter[]) Array of Parameter objects that correlate to Input Parameters in Rule Sets.</td></tr>
</table>

	Used by ExecuteIndependentRuleSet and ExecuteRuleSet.

Child Members

- RuleApp
- RuleEngineServiceOptions
- RuleEngineServiceOutputTypes

4.7.2.2.1 Parameter

Member Elements

Name	Description
Name	The name of the Input Parameter of the Rule Set
Value	The value of the Input Parameter of the Rule Set

4.7.2.2.2 RuleApp

Member Elements

Name	Description
CacheTimeout	(int) The minimum rule application refresh interval (in seconds) that specifies when to check the catalog for an updated revision, if latest revision or label was specified. Used only for Catalog Rule Applications.
ConnTimeout	(int) The Catalog Service connection timeout (in seconds). If the timeout expires without a successful connection to irCatalog, an exception will be thrown. The default is 60 seconds. Used only for Catalog Rule Applications.
FileName	(string) File name of File System Rule Application. The file must be located in the configured rule application folder on the server. Used only for Catalog Rule Applications.
Password	(string) Password used to authenticate to irCatalog. If **UserName** and **Password** are not provided irServer will use the username and password provided in the configuration file. Used only for Catalog Rule Applications.
RepositoryRuleAppRevisionSpec	Parent element containing information used to retrieve rule applications from irCatalog. Used only for Catalog Rule Applications.
RepositoryServiceUrl	(string) Url to irCatalog. If **RepositoryServiceUrl** is not provided, irServer will use the repository url provided in the configuration file. Only used for Catalog Rule Applications.

UseIntegratedSecurity	(boolean) True to have irServer authenticate to irCatalog via integrated security, otherwise use the provided **UserName** and **Password**. Used only for Catalog Rule Applications.
UserName	(string) UserName used to authenticate to irCatalog. If **UserName** and **Password** are not provided irServer will use the username and password provided in the configuration file. Used only for Catalog Rule Applications.

Child Members

- RepositoryRuleAppRevisionSpec

4.7.2.2.2.1 RepositoryRuleAppRevisionSpec

Name	Description
Guid	(Guid) The GUID of the Rule Application. If specified, the value will be used to locate the Rule Application in irCatalog. If omitted, **RuleApplicationName** must be specified.
Label	(string) The label of the revision to retrieve. If specified, the value will be used to locate the Rule Application in irCatalog. If both **Label** and **Revision** are omitted, the latest version of the rule application will be retrieved.
Revision	(int) The revision number to retrieve. If specified, the value will be used to locate the Rule Application in irCatalog. If both **Label** and **Revision** are omitted, the latest version of the rule application will be retrieved.
RuleApplicationName	(string) The name of the Rule Application. If specified, the value will be used to locate the Rule Application in irCatalog. If omitted, **Guid** must be specified.

4.7.2.2.3 RuleEngineServiceOptions

Member Elements

Name	Description
Overrides	Parent container for Endpoint and Data Element overrides. See Overrides for more information.
RuleSessionOverrides	Parent element for Rule Session Overrides. Allows you to specify overrides for **MaxEvaluationCyclesOverride**, **ExecutionTimeoutOverride** and **CurrentDateOverride**.

4.7.2.2.4 RuleEngineServiceOutputTypes

Member Elements

Name	Description
ActiveNotifications	(boolean) True to return notifications, otherwise notifications will be omitted from the response. Defaults to **true** if not specified.
ActiveValidations	(boolean) True to return validations, otherwise validations will be omitted from the response. Defaults to **true** if not specified.
EntityState	(boolean) True to return entity state, otherwise entity state will be omitted from the response. Defaults to **true** if not specified.
Overrides	(boolean) True to return overrides, otherwise overrides will be omitted from the response. Defaults to **false** if not specified.
RuleExecutionLog	(boolean) True to return the rule execution log, otherwise the rule execution log will be omitted from the response. Defaults to **false** if not specified. If there are runtime errors encountered during execution, the rule execution log will always be returned in the response.

4.7.2.3 Overriding RuleApp Endpoints at Runtime

Supported Overrides

- Database Connection String
- Mail Server Connection
- Web Server Address
- Web Service WSDL Uri
- Web Service MaxReceivedMessageSize
- XML Document Path
- XML Schema
- XML Schema Validation
- Inline Table
- Inline XML Document
- Inline Value List
- SQL Query
- REST Service X.509 Certificate Path
- REST Service Authentication Type

- REST Service Root URL

4.7.2.3.1 Database Connection String

Note: Be sure to specify the ConnectionString override first.

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <ConnectionString>new connection string</ConnectionString>
        <OverrideType>DatabaseConnection</OverrideType>
        <Name>endpoint name</Name>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
   "RuleEngineServiceOptions":{
         "Overrides":[
          {
             "ConnectionString":"new connection string",
             "OverrideType":"DatabaseConnection",
             "Name": "endpoint name"
          }]
   }
}
```

4.7.2.3.2 Mail Server Connection

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <OverrideType>MailServerConnection</OverrideType>
        <Name>endpoint name</Name>
        <ServerAddress>mail server</ServerAddress>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
   "RuleEngineServiceOptions":{
         "Overrides":[
          {
             "OverrideType":"MailServerConnection",
             "Name":"endpoint name",
             "ServerAddress":"mail server"
          }]
   }
}
```

4.7.2.3.3 Web Service Address

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <OverrideType>WebServiceAddress</OverrideType>
        <Name>endpoint name</Name>
        <ServiceUriOverride>mail server</ServiceUriOverride>
        </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
   "RuleEngineServiceOptions":{
         "Overrides":[
          {
             "OverrideType":"WebServiceAddress",
             "Name":"endpoint name",
             "ServiceUriOverride":"mail server"
          }]
   }
}
```

4.7.2.3.4 Web Service WSDL Uri

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <OverrideType>WebServiceWsdlUri</OverrideType>
        <Name>endpoint name</Name>
        <WsdlUri>wsdl uri</WsdlUri>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
   "RuleEngineServiceOptions":{
         "Overrides":[
          {
             "OverrideType":"WebServiceWsdlUri",
             "Name":"endpoint name",
             "WsdlUri":"wsdl uri"
          }]
   }
}
```

4.7.2.3.5 Web Service MaxReceivedMessageSize

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <OverrideType>WebServiceMaxReceivedMessageSize</OverrideType>
        <Name>endpoint name</Name>
        <WebServiceMaxReceivedMessageSize>message size in bytes</WebServiceMaxReceivedMessageSize>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
   "RuleEngineServiceOptions":{
         "Overrides": [
          {
             "OverrideType": "WebServiceMaxReceivedMessageSize",
             "Name": "endpoint name",
             "WebServiceMaxReceivedMessageSize": message size in bytes
          }]
   }
}
```

4.7.2.3.6 XML Document Path

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <OverrideType>XmlDocumentPath</OverrideType>
        <Name>endpoint name</Name>
        <XmlPath>xml path</XmlPath>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
   "RuleEngineServiceOptions":{
         "Overrides":[
          {
             "OverrideType":"XmlDocumentPath",
             "Name":"endpoint name",
             "XmlPath":"xml path"
          }]
   }
}
```

4.7.2.3.7 XML Schema

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <OverrideType>XmlSchema</OverrideType>
        <Name>endpoint name</Name>
        <XsdPath>xsd path</XsdPath>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
    "RuleEngineServiceOptions":{
          "Overrides":[
           {
               "OverrideType":"XmlSchema",
               "Name":"endpoint name",
               "XsdPath":"xsd path"
           }]
    }
}
```

4.7.2.3.8 XML Schema Validation

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <OverrideType>XmlSchemaValidation</OverrideType>
        <Name>endpoint name</Name>
        <EnableXsdValidation>true/false</EnableXsdValidation>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
    "RuleEngineServiceOptions":{
          "Overrides":[
           {
               "OverrideType":"XmlSchemaValidation",
               "Name":"endpoint name",
               "EnableXsdValidation":"true/false"
           }]
    }
}
```

4.7.2.3.9 Inline Table

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
```

```
      <Override>
        <OverrideType>InlineTable</OverrideType>
        <Name>endpoint name</Name>
        <TableSettings>inline table</TableSettings>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
   "RuleEngineServiceOptions":{
         "Overrides":[
          {
             "OverrideType":"InlineTable",
             "Name":"endpoint name",
             "TableSettings":"inline table"
          }]
   }
}
```

4.7.2.3.10 Inline XML Document

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <OverrideType>InlineXmlDocument</OverrideType>
        <Name>endpoint name</Name>
        <Settings>settings</Settings>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
   "RuleEngineServiceOptions":{
         "Overrides":[
          {
             "OverrideType":"InlineXmlDocument",
             "Name":"endpoint name",
             "Settings":"settings"
          }]
   }
}
```

4.7.2.3.11 Inline Value List

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <OverrideType>InlineValueList</OverrideType>
        <Name>endpoint name</Name>
        <ValueListItems>
           <ValueListItem>
             <DisplayText>display text</DisplayText>
             <Value>value</Value>
           </ValueListItem>
       </ValueListItems>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
   "RuleEngineServiceOptions":{
         "Overrides":[
          {
             "OverrideType":"InlineValueList",
             "Name":"endpoint name",
             "ValueListItems":[{
                 "DisplayText": "display text",
                 "Value": "value"
             }]
          }]
   }
}
```

4.7.2.3.12 SQL Query

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <OverrideType>SqlQuery</OverrideType>
        <Name>endpoint name</Name>
        <Query>query</Query>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
   "RuleEngineServiceOptions":{
         "Overrides":[
          {
             "OverrideType":"SqlQuery",
             "Name":"endpoint name",
             "Query":"query"
          }]
   }
}
```

4.7.2.3.13 REST Service X.509 Certificate Path

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <OverrideType>RestServiceX509CertificatePath</OverrideType>
        <Name>endpoint name</Name>
        <RestServiceX509CertificatePassword>new connection string</RestServiceX509CertificatePassw
        <RestServiceX509CertificatePath>new connection string</RestServiceX509CertificatePath>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
   "RuleEngineServiceOptions":{
         "Overrides":[
          {
             "OverrideType":"RestServiceX509CertificatePath",
             "Name" : "endpoint name",
             "RestServiceX509CertificatePassword": "certificate password",
             "RestServiceX509CertificatePath": "path to cert"
          }]
   }
}
```

4.7.2.3.14 REST Service Authentication Type

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <AuthenticationType>None/Basic/NTLM/Kerberos/Custom</AuthenticationType>
        <Name>endpoint name</Name>
        <OverrideType>RestServiceAuthenticationType</OverrideType>
        <RestServiceDomain>domain</RestServiceDomain>
        <RestServicePassword>password</RestServicePassword>
        <RestServiceUserName>userName</RestServiceUserName>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
   "RuleEngineServiceOptions":{
         "Overrides":[
          {
             "AuthenticationType": "None/Basic/NTLM/Kerberos/Custom",
             "Name": "endpoint name",
             "OverrideType": "RestServiceAuthenticationType",
             "RestServiceDomain": "domain",
             "RestServicePassword": "password",
             "RestServiceUserName": "userName"
          }]
   }
}
```

4.7.2.3.15 REST Service Root Url

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <Name>endpoint name</Name>
        <OverrideType>RestServiceRootUrl</OverrideType>
        <RestServiceRootUrl>new url</RestServiceRootUrl>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
   "RuleEngineServiceOptions":{
         "Overrides":[
          {
             "Name": "endpoint name",
             "OverrideType": "RestServiceRootUrl",
             "RestServiceRootUrl": "RestServiceRootUrl"
          }]
   }
}
```

4.7.2.3.16 REST Service Allow Untrusted Certificate

Sample XML

```
<Request>
...
  <RuleEngineServiceOptions>
    <Overrides>
      <Override>
        <Name>endpoint name</Name>
        <OverrideType>RestServiceAllowUntrustedCertificates</OverrideType>
        <AllowUntrustedCertificates>true</AllowUnTrustedCertificates>
      </Override>
    </Overrides>
  ...
  </RuleEngineServiceOptions>
...
</Request>
```

Sample JSON

```
{
   "RuleEngineServiceOptions":{
       "Overrides":[
       {
              "Name": "endpoint name",
              "OverrideType": "RestServiceAllowUntrustedCertificates",
              "AllowUnTrustedCertificates": "true"
       }]
   }
}
```

4.7.2.4 Enabling Web-based help

Enabling Web-based help

The WCF auto-generated help page, with details of the REST execution service, can now be enabled and made available in the browser via the service URI.
These details include the available operations of the service and help provide confirmation that the service is installed and responsive.

To enable the help page for the REST execution service, modify the endpoint behavior in the Rule Execution Service (Rule Engine Service) configuration file from:

```
<behavior name="ContentNegotiation">
          <webHttp automaticFormatSelectionEnabled="true" />
</behavior>
```

to:

```
<behavior name="ContentNegotiation">
          <webHttp helpEnabled="true" automaticFormatSelectionEnabled="true" />
</behavior>
```

4.7.2.5 Source Code Examples

The following code samples have been provided for accessing the irServer - Rule Execution Service via REST.

- Sychronously calling the REST Service
- Asychronously calling the REST Service with JSON
- Asychronously calling the REST Service with XML

4.8 Calling InRule from BizTalk Server

irAdapter for BizTalk Server provides three ways to interact with an InRule installation:

- Calling InRule from a BizTalk Orchestration
- irAdapter for BizTalk Server
- Static .NET methods that may be called from BizTalk Message Assignment shapes

4.8.1 Calling InRule from a BizTalk Orchestration

Here are some suggested configurations for calling InRule from BizTalk.

Calling InRule from within a BizTalk Orchestration using Message Assignment

For low-load scenarios, InRule can be invoked directly from a BizTalk orchestration using the Message Assignment shape. This approach is not recommended for the following scenarios:

- Orchestrations that will experience high sustained loads
- Rule applications that have a runtime of greater than 500 ms per request

Calling InRule using a send-receive port and the irAdapter for BizTalk Server

For higher load scenarios, the irAdapter for BizTalk Server can be used with a send-receive port. The adapter allows for more flexible hosting options to be used with BizTalk that control load

Calling InRule using a send-receive port and a custom BizTalk adpater

In high-load scenarios where specialized calls must be made to irSDK, a custom BizTalk adapter can be used with a send-receive port. Like the irAdapter, this scenario allows for more flexible hosting and throttling options with built-in BizTalk configuration settings. Please contact InRule support for sample code to create a custom BizTalk adapter with irSDK.

Hosting irAdapter and other custom adapters

The suggested configuration for hosting BizTalk adapaters with InRule is as follows:

- Host irAdapter or the custom adapter in a separate BizTalk Host instance
- Configure the MaxWorkThreads for the separate host instance to either four threads or one thread per processing core, whichever is greater. Please see the following link for more information https://msdn.microsoft.com/en-us/library/aa561380.aspx

4.8.2 Using irAdapter for BizTalk Server

Configuring irAdapter in BizTalk

irAdapter for BizTalk Server must first be added to the Adapters list in the BizTalk Server Administration tool.

1. Launch BizTalk Server Administrator and navigate to the following node:

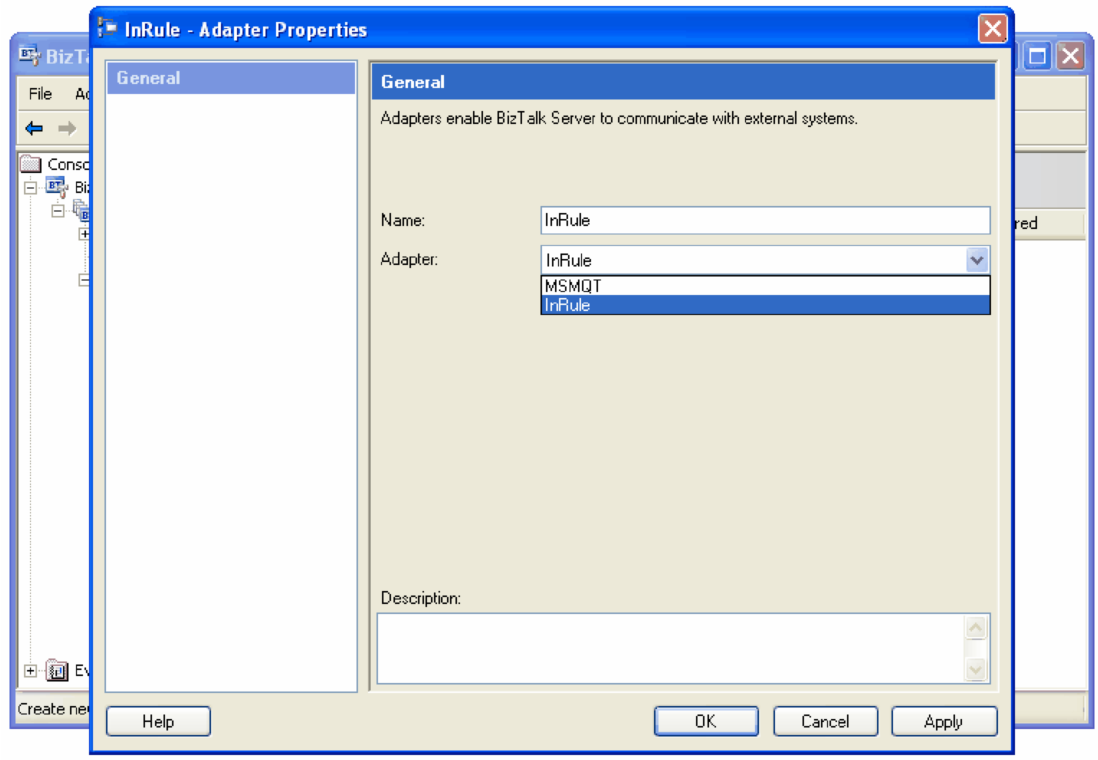
   * BizTalk Server Administration

   * BizTalk Group

   * Platform Settings

   * Adapters

2. Right-click Adapters and select New > Adapter, then name it InRule and choose InRule from the dropdown list as shown here.



InRule - Adapter Properties

General

Adapters enable BizTalk Server to communicate with external systems.

Name: InRule

Adapter: InRule

MSMQT

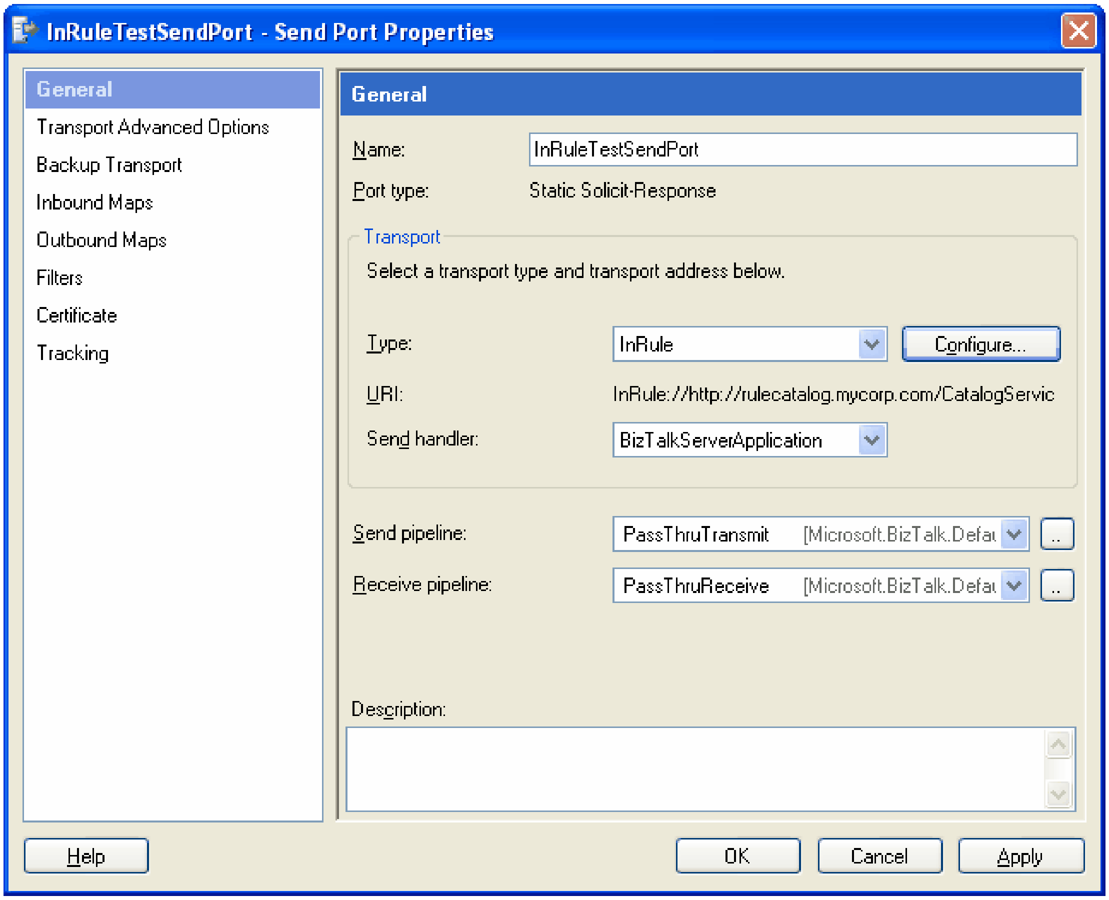
InRule

Description:

Using irAdapter - Creating and configuring a send port

1. Launch the BizTalk Server Administrative Console.

2. Right-click on the BizTalk application’s Send Ports node and select New, then “Static One-way Send Port” or “Static Solicit-Response Send Port”. The latter is required if a response is desired from the rule engine; however, scenarios do exist where a response is not required. Select Solicit-Response if you don’t know which to choose.
3. Name the adapter and select InRule as the transport type:



InRuleTestSendPort - Send Port Properties

General

Name: InRuleTestSendPort

Port type: Static Solicit-Response

Transport

Select a transport type and transport address below.

Type: InRule

URI: InRule://http://rulecatalog.mycorp.com/CatalogServic

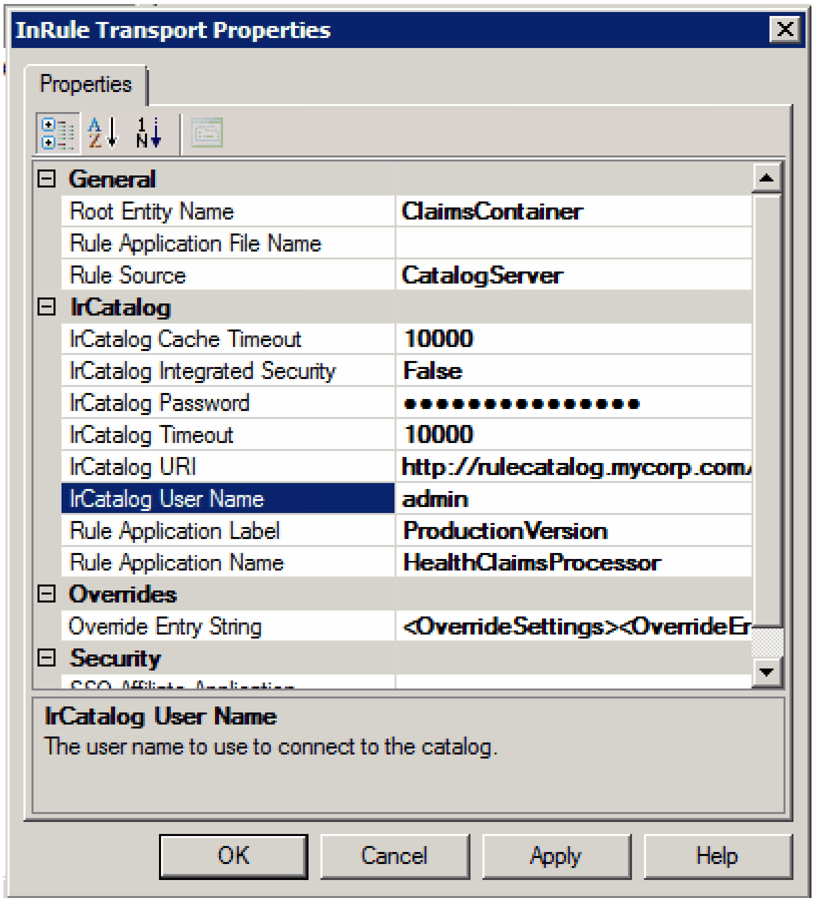
Send handler: BizTalkServerApplication

Send pipeline: PassThruTransmit [Microsoft.BizTalk.Defau

Receive pipeline: PassThruReceive [Microsoft.BizTalk.Defau

Description:

4. Next, click Configure:



General	
Root Entity Name	ClaimsContainer
Rule Application File Name	
Rule Source	CatalogServer
IrCatalog	
IrCatalog Cache Timeout	10000
IrCatalog Integrated Security	False
IrCatalog Password	●●●●●●●●●●●●●●●
IrCatalog Timeout	10000
IrCatalog URI	http://rulecatalog.mycorp.com/
IrCatalog User Name	admin
Rule Application Label	ProductionVersion
Rule Application Name	HealthClaimsProcessor
Overrides	
Override Entry String	<OverrideSettings><OverrideEr
Security	
SSO Affiliate Application	

IrCatalog User Name
The user name to use to connect to the catalog.

5. Specify the Root Entity Name. This is the entity that will be populated with the XML message sent to the adapter. For example, if a rule application contained the entities Invoice, LineItem and Customer, and Invoice was the "top level" entity, "Invoice" would be the correct value. (The Root Entity Name is typically the same as the root element name in the XML sent to the adapter.)
6. Set Rule Source to CatalogServer if the rule application will be retrieved from a catalog server, or File if the rule application will be loaded from disk.
7. If CatalogServer is selected:
   a. Set irCatalog Integrated Security to True if the identity of the process hosting the BizTalk instance should be used to authenticate against the catalog. Otherwise, set it to False and provide values for irCatalog User Name and irCatalog Password.
   b. Set Rule Application Name to the name of the rule application that will be retrieved.
   c. Optionally set Rule Application Label to the label of the rule application specified in Rule Application Name.
   d. Set irCatalog URI to the URI of the catalog server. (The URI can be found by running the Configuration Utility in the InRule start menu group.)
   e. Change the values of the irCatalog timeout and/or Cache Timeout. The former is the connection to the server and the latter specifies the interval at which the adapter will check for new versions of the rule application on the catalog server. Both of these numbers should be defined and set to values larger than zero.
8. If File is selected, set Rule Application File Name.
9. If the Rule Application End Points need to be overridden in the given environment, then an XML string can be provided to define the overrides. In the the case of Database Connections and Mail Servers, and single Value should be defined for the OverrideEntry. For Web Service

End Points, at least two Values are required -- one for the service URI and one for the service WSDL. Web Service End Points also support an optional third Value for the location of the X509 certificate.

A sample Override Entry String XML that overrides a database and a web service end point:

```
<OverrideSettings>
  <OverrideEntries>
    <OverrideEntry>
      <Name>DemoDB</Name>
      <Value>My Connection String</Value>
    </OverrideEntry>
    <OverrideEntry>
      <Name>DemoWebService</Name>
      <Value>http://myservice.mycorp.com/service.svc/method</Value>
      <Value>http://myservice.mycorp.com/service.svc?wsdl</Value>
    </OverrideEntry>
  </OverrideEntries>
</OverrideSettings>
```

4.8.3 Using the BizTalk Message Assignment Shape

Static methods available to BizTalk expression shapes

The InRule.BizTalk.Orchestration.RuleExecution class contains the following methods that may be called from a BizTalk Expression: ExecuteRulesFromCatalog and ExecuteRulesFromFile. Both return an XML document with rule results.

ExecuteRulesFromCatalog - contains the following parameters:

- message: The message (single part) or message part containing the XML to be passed to the rule engine.
- outMessage: The message or message part that should contain the updated message.
- rootEntityName: The name of the “top level” entity that will be populated with the XML in the “message” parameter.
- ruleApplicationLabel: The optional label for the application.
- ruleApplicationName: The application name on the catalog server.
- useIntegratedSecurity: Indicates whether integrated security should be used.
- username: The user name to use to authenticate to the server.
- password: The password to use to authenticate to the server.
- catalogUri: The uri of the server.
- catalogTimeout: The amount of time, in milliseconds, to wait for the catalog server.
- catalogCacheTimeout: The amount of time, in milliseconds, the adapter will cache the retrieved rule application.

ExecuteRulesFromFile - contains the following parameters:

- message: The message (single part) or message part containing the XML to be passed to the rule engine.

- outMessage: The message or message part that should contain the updated message.
- rootEntityName: The name of the "top level" entity that will be populated with the XML in the "message" parameter.
- filePath: The file system path to the rule application file. Note that any relative paths will be relative to the BizTalk installation directory.

4.9 Calling InRule from Windows Workflow Foundation

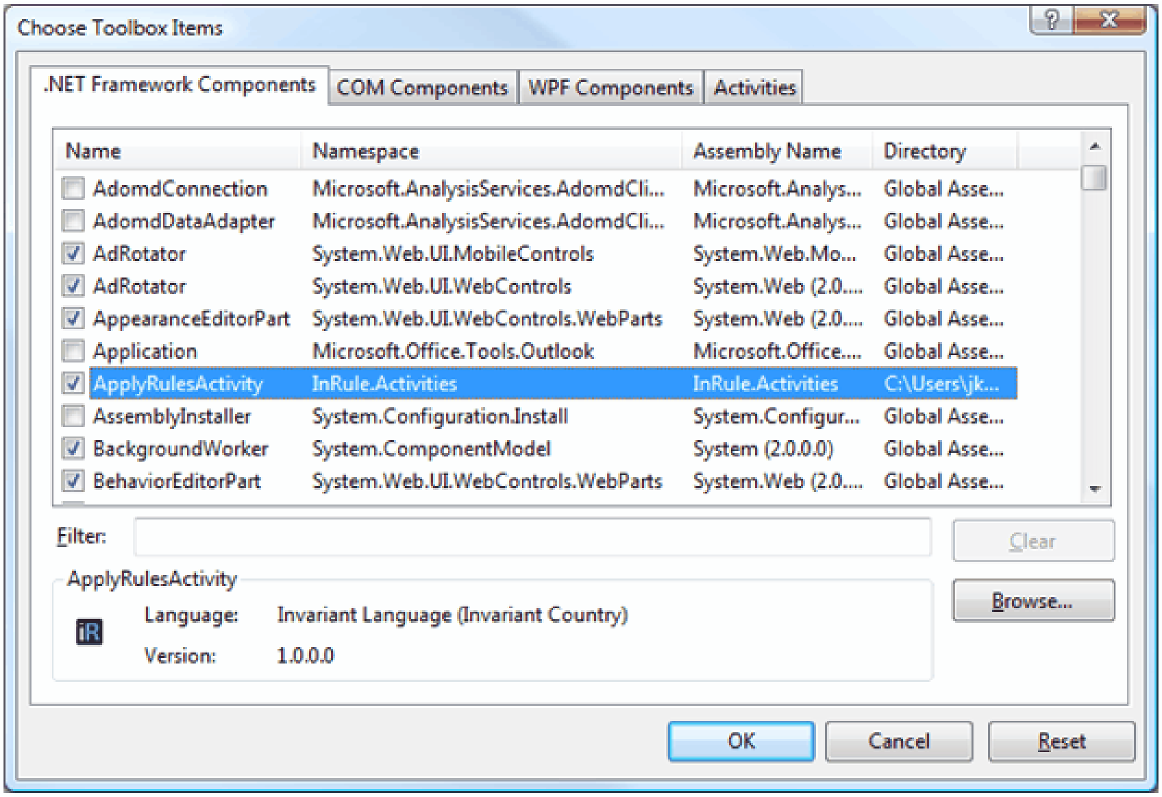
InRule comes installed with a custom activity for integrating calls to the rule engine from Windows Workflow applications.

- Installing the ApplyRules Windows Workflow Activity in Visual Studio
- Configuration of the ApplyRules Windows Workflow Activity
- Rule Execution Log Service

4.9.1 Installing the Activity

To install the Apply Rules WF activity in Microsoft Visual Studio:

1. Open a project containing a WF workflow and open the workflow in the workflow designer.
2. Right-click the Toolbox and select "Choose Items…".
3. Click the Browse button and navigate to the <InRule installation directory>\irSDK\bin folder.
4. Select the InRule.Activities.dll assembly and click Open.
5. Confirm that the ApplyRulesActivity is selected and click OK.



Name	Namespace	Assembly Name	Directory
☐ AdomdConnection	Microsoft.AnalysisServices.AdomdCli...	Microsoft.Analys...	Global Asse...
☐ AdomdDataAdapter	Microsoft.AnalysisServices.AdomdCli...	Microsoft.Analys...	Global Asse...
☑ AdRotator	System.Web.UI.MobileControls	System.Web.Mo...	Global Asse...
☑ AdRotator	System.Web.UI.WebControls	System.Web (2.0....	Global Asse...
☑ AppearanceEditorPart	System.Web.UI.WebControls.WebParts	System.Web (2.0....	Global Asse...
☐ Application	Microsoft.Office.Tools.Outlook	Microsoft.Office....	Global Asse...
☑ ApplyRulesActivity	InRule.Activities	InRule.Activities	C:\Users\jk...
☐ AssemblyInstaller	System.Configuration.Install	System.Configur...	Global Asse...
☑ BackgroundWorker	System.ComponentModel	System (2.0.0.0)	Global Asse...
☑ BehaviorEditorPart	System.Web.UI.WebControls.WebParts	System.Web (2.0....	Global Asse...

Filter:

ApplyRulesActivity

Language: Invariant Language (Invariant Country)

Version: 1.0.0.0

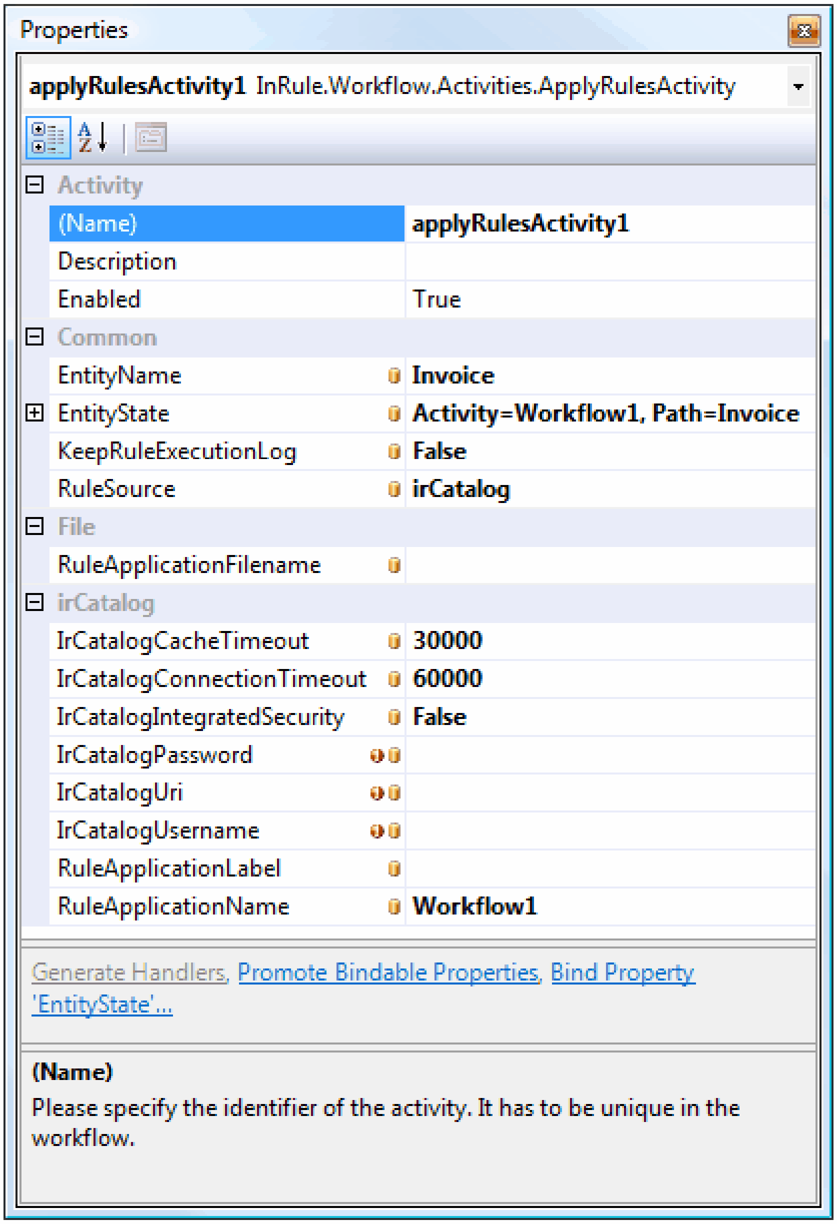
4.9.2 Configuration of the Activity

The following information details the steps required for configuration of the InRule ApplyRules activity for Windows Workflow.

Overview

The ApplyRules activity contains three groups of properties:

- **irCatalog:** Those that are required when executing rules retrieved from an irCatalog
- **File:** A filename, required when executing rules from a file-base rule application
- **Common:** Settings required when using either of the above



Properties

applyRulesActivity1 InRule.Workflow.Activities.ApplyRulesActivity

⊟ Activity	
(Name)	applyRulesActivity1
Description	
Enabled	True
⊟ Common	
EntityName	Invoice
⊞ EntityState	Activity=Workflow1, Path=Invoice
KeepRuleExecutionLog	False
RuleSource	irCatalog
⊟ File	
RuleApplicationFilename	
⊟ irCatalog	
IrCatalogCacheTimeout	30000
IrCatalogConnectionTimeout	60000
IrCatalogIntegratedSecurity	False
IrCatalogPassword	
IrCatalogUri	
IrCatalogUsername	
RuleApplicationLabel	
RuleApplicationName	Workflow1

Generate Handlers, Promote Bindable Properties, Bind Property 'EntityState'...

(Name)
Please specify the identifier of the activity. It has to be unique in the workflow.

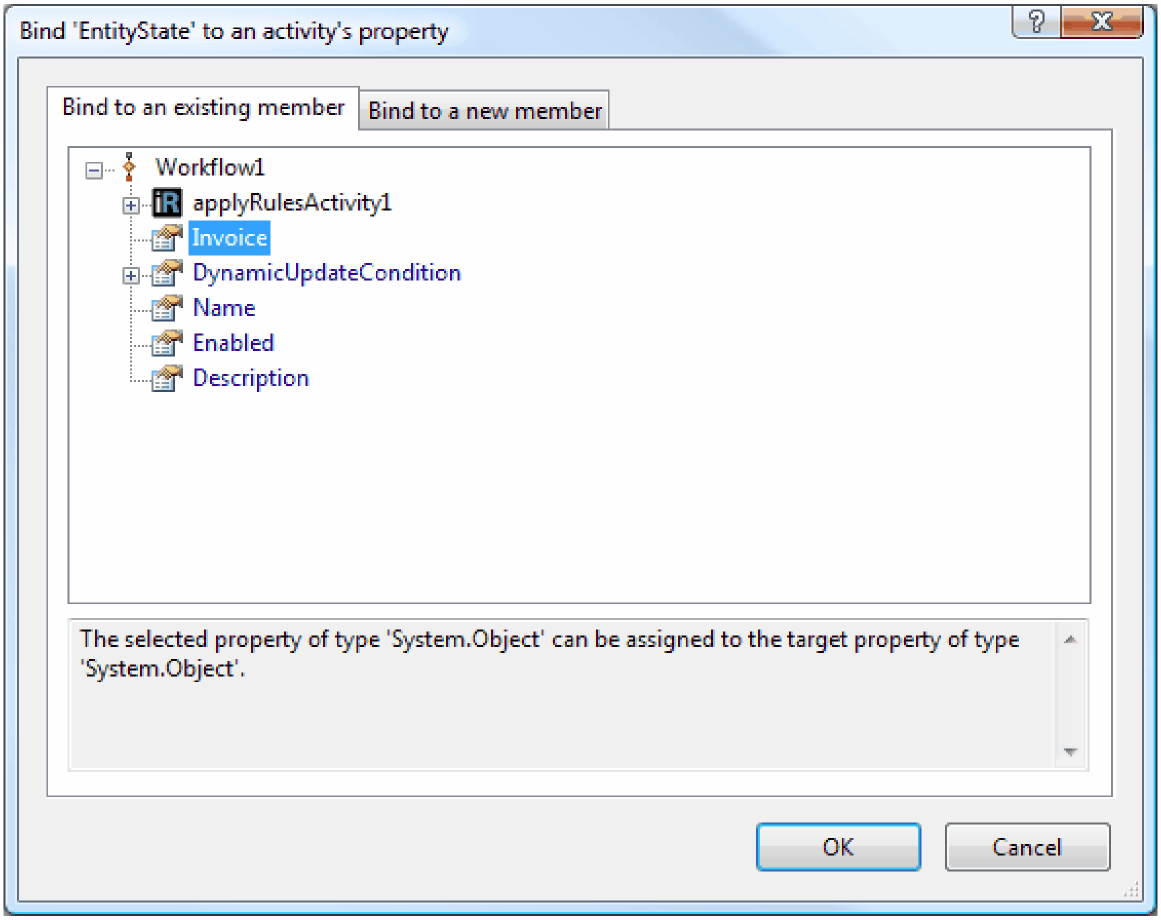
Figure 1: Apply Rules activity property grid

The Common Group

All of the following properties are required:

Entity Name	The name of the entity that will be created.
EntityState	The object that the rule engine will execute rules against. Figure 1 shows a property grid that indicates that an Invoice entity will be created, and the object returned by the workflow's Invoice property will be used as the entity state. Note that EntityState is a dependency property, so it's easy to select the entity state by clicking the EntityState cell, then the ellipsis button. Figure 2 shows

	the dependency property binding form.
KeepRuleExecutionLog	Indicates whether the rule execution log will be saved after rule execution ends. If set to true, the log can be accessed via the LastRuleExecutionLog property on the activity.
Rule Source	This setting indicates whether the rules will be retrieved from an irCatalog or the file system.



The selected property of type 'System.Object' can be assigned to the target property of type 'System.Object'.

Figure 2: EntityState selection dialog.

The File Group

This property is only required when RuleSource is set to RuleApplicationFile.

Filename	The filename of the rule application file (*.ruleapp).

The irCatalog Group

These properties are required when RuleSource is set to irCatalog.

IrCatalogCacheTimeout	The number of milliseconds to cache a rule application retrieve from an irCatalog.
IrCatalogConnectionTimeout	The number of milliseconds to wait for a response from the irCatalog.
IrCatalogIntegratedSecurity	Indicates whether integrated security should be used instead of a username and password.

IrCatalogPassword	The password for the irCatalog. Required only when IrCatalogIntegratedSecurity is false.
IrCatalogUri	The URI of the irCatalog.
IrCatalogUsername	The username for the irCatalog. Required only when IrCatalogIntegratedSecurity is false.
RuleApplicationLabel	The label of the rule application to retrieve. If a label is not specified, the latest version is retrieved.
RuleApplicationName	The name of the rule application to retrieve. This value is required.

4.9.3 Rule Execution Log Service

Overview

The Rule Execution Log Service allows application-level access to rule execution logs created during workflow execution.

At a high level, the process is as follows:

1. A RuleExecutionLogService object is created and populated with the names of the ApplyRules activities it will monitor.
2. The workflow is executed. Every time an ApplyRules activity is executed, the service checks to see if it is one that is being monitored. If so, its execution log is saved.
3. After execution, the execution logs can be retrieved from the service and processed.

Creating and Populating the Service

To create and populate the service, simply create a RuleExecutionLogService object and populate its ActivityNames collection with the names of the activities to monitor. Once all activity names have been added, add the service to the workflow runtime via its AddService method.

```
using (WorkflowRuntime workflowRuntime = new WorkflowRuntime())
{
    RuleExecutionLogService logService = new RuleExecutionLogService();
    logService.ActivityNames.Add("applyRules");
    workflowRuntime.AddService(logService);
```

Retrieving the Execution Logs

The rule execution log(s) may be retrieved during or after workflow execution. First, a reference to the service is required. This reference may already exist if the service was created in the same method; otherwise, a reference may be obtained via WorkflowRuntime's GetService method. Once a reference has been obtained, a call to the GetExecutionLogs method, with the name of the activity as an argument, will return a collection of rule execution logs. The collection contains a rule execution log for every time it was executed during the course of the workflow.

```
    List<RuleExecutionLog> logs = logService.GetExecutionLogs("applyRules");
    RuleExecutionLog log = logs[0];
    Console.WriteLine("Total execution time: {0:n3} ms",
    log.TotalExecutionTime);
}
```

Difference between service and ApplyRules activity's KeepRuleExecutionLog and LastRuleExecutionLog

The service's primary function is to serve as a way to access rule execution logs from outside the context of an executing workflow. The ApplyRules activity's KeepRuleExecutionLog property specifies whether the most recent rule execution log will be kept and available via its LastExecutionLog property, which may then be accessed by anything participating in the workflow. It is in no way related to the service.

Guidelines are as follows:

- To access a rule execution log from within a workflow, use ApplyRules' KeepRuleExecutionLog/LastRuleExecutionLog
- To access one or more rule execution logs from outside the workflow, use the RuleExecutionLogService.

4.10 .NET Assembly Object State

This feature allows InRule Entities and Fields to be bound to .NET objects at runtime.

The benefit is that state can be read and written directly from/to objects instead of explicitly setting individual Field values via irSDK or using JSON/XML serialization mechanisms.

The Rule Application schema may be imported from various sources at design-time (authoring). When imported from a .NET Assembly, the classes and fields/properties govern the Entity/Field names and structure of the InRule schema and will attempt synchronize any changes to the underlying Assembly.

At runtime, the default behavior of the Rule Session will create Entities that are bound to object instances of the classes imported by the .NET Assembly Schema during authoring.

- Differences in Assembly Binding Behavior at Runtime vs Authoring-time
- Binding to Collections
- .NET Assembly State Refresh Options

4.10.1 Differences in Assembly Binding Behavior at Runtime vs Authoring-time

Authoring

As part of the rule application development process, authors may choose to make use of existing class libraries and data definitions.

Using a .NET Assembly Schema, classes may be imported as InRule Entities, along with their associated fields/properties representing InRule Fields, and methods that may be called by rules.

Like any schema type, .NET Assembly Schemas use the associated resource as the source for binding to a Rule Application's schema. While the Schema is in place, no design-time modifications to the Rule Applications's schema are allowed. Temporary Fields and Collections can still be added for data that does not need to be exposed outside of the rules.

Runtime

At runtime, any Rule Application that directly executes rules bound to a .NET Assembly Schema will need to have that .NET Assembly .dll (and any required dependencies) deployed along with it.

Code integrating with irSDK to execute rules will create an instance of an InRule.Runtime.Entity. This Entity will be bound to an instance of the class that was imported at design-time.

This is the default behavior for InRule schemas bound to .NET Assembly Schemas at design time.

Entities may optionally be bound to arbitrary objects whose fields/properties match the Entity Field/ Collection names (duck typing), regardless of whether the schema was bound to a .NET Assembly Schema at design-time.

Design Time Schema	Default Runtime Binding Behavior	Optional Runtime Binding Behavior
Unbound	Unbound	Arbitrary object, XElement, Dictionary<string, object>, ExpandoObject
.NET Assembly Schema	Bound to classes configured at design-time	Arbitrary object, XElement, Dictionary<string, object>, ExpandoObject
Database Schema	Unbound	Arbitrary object, XElement, Dictionary<string, object>, ExpandoObject
XML Schema	Unbound	Arbitrary object, XElement, Dictionary<string, object>, ExpandoObject

In addition to arbitrary objects, the following .NET Framework types are special-cased when explicitly passed to irSDK at runtime:

.NET Framework Type	Notes
XElement	Child XElements will map to InRule Fields by name, and the content of each child XElement will become the Field value.
Dictionary<string, object>	For each item in the Dictionary, the key must match a Field name of the bound Entity. The value of each item should contain a primitive value or object value that matches the associated Field type. Even though a Dictionary implements ICollection<T>, it is not treated as an InRule Collection.
ExpandoObject	Behaves similarly to Dictionary where the dynamic property names should match the InRule Field names.

Examples

Default:

```
var invoice = session.CreateEntity("Invoice");
```

If an object is not explicitly passed to the CreateEntity() method, an instance of the the class configured at design-time will be bound to the Entity if the source is a .NET Assembly Schema. For all other schema types, the Entity will be unbound.

Populated Configured Object:

```
var invoice = session.CreateEntity("Invoice", new Invoice() { InvoiceDate = new
DateTime(2019, 1, 1) });
```

An instance of the class is populated and explicitly passed to the CreateEntity() method. It does not matter whether the Invoice class was imported from a .NET Assembly Schema or not. If it was not, then the object's fields/properties must match the Field names on the Entity.

Arbitrary Object:

```
var invoice = session.CreateEntity("Invoice", new LooksLikeInvoice() { InvoiceDate
= new DateTime(2019, 1, 1) });
```

An arbitrary object that exhibits the same properties as the Invoice Enitity is explicitly passed to the CreateEntity() method.

XElement:

```
var invoice = session.CreateEntity(“Invoice”,
XElement.Parse(“<Invoice><InvoiceDate>2019-01-01T00:00:00</InvoiceDate></
Invoice>”));
```

XElement is created and explicitly passed to the CreateEntity() method.
Note: This should not be confused with passing XML as a string to the CreateEntity() method, as this does not bind the Entity to any object.

Dictionary<string, object>:

```
var invoice = session.CreateEntity(“Invoice”, new Dictionary<string, object>
{ { “InvoiceDate”, new DateTime(2019, 1, 1) }, {“LineItems”, new
List<LineItem>()} };
```

Dictionary instance is initialized with the InRule Field names and associated values and explicitly passed to the CreateEntity() method.

ExpandoObject:

```
dynamic obj = new ExpandoObject();
obj.InvoiceDate = new DateTime(2019, 1, 1);
var invoice = session.CreateEntity(“Invoice”, obj);
```

ExpandoObject is populated with properties matching Field names on the Entity and explicitly passed to the CreateEntity() method.

4.10.2 Binding to Collections

Historically, an object exposing collection properties required the .NET collection type to implement either IList or IList<T> in order to be used in a .NET Assembly Schema binding.

With the release of InRule v5.3.0, support for collection types in assembly schemas has been expanded. The following collection types are now supported (with corresponding feature sets):

	Add Member	Remove Member	Clear	Sort	Get Member by Index	Set Member by Index
IList	Yes*	Yes	Yes	Yes	Yes	Yes
IList<T>	Yes	Yes	Yes	Yes	Yes	Yes
ICollection<T>	Yes	Yes**	Yes	No	Yes***	No
IEnumerable<T>	No	No	No	No	Yes***	No
Arrays	No	No	No	Yes	Yes	Yes

* Type of collection property may implement IList but cannot be declared as IList because InRule cannot infer the type of the collection member to instantiate.

** ICollection<T> collections can only remove items by object, not by index, therefore this only works if the member object at the index being removed does not exist in the same collection more than once; if it does, an Exception will be thrown because the collection would only remove the first instance of the object in the collection, which may not be the desired index.

*** ICollection<T> and IEnumerable<T> do not ensure deterministic ordering of items that they contain, however most implementations tend to enumerate the items in the same order each time.

Addressing a specific index in these collections is emulated by advancing the enumerator the desired number of times to access the member by index.

4.10.3 .NET Assembly State Refresh Options

When a rule application schema is bound to a .NET assembly and the instance methods from the bound classes are being called from within rules, there is a setting available that can improve performance in certain situations. By default, all bound fields must be refreshed in order to ensure the dependency network is aware of the changes that occurred within the method. Minimizing the fields that are refreshed after the call can reduce overall processing time, especially for large bound schemas or when instance methods are called often. The following directions will explain the process of telling the rules engine to only update specific fields specified by the user.

By double-clicking the method name in the .NET assembly schema import pane, a "RuleWrite Information" dialog appears which allows the user to control whether or not the method call action in InRule needs to refresh the working memory state with the updated object memory state. The setting "Enable full refresh of all bound objects "(which is enabled by default) will refresh the entire object state to working memory after each method invocation. This setting may be unchecked when it is known that the method will not result in underlying state changes that need to be refreshed. In situations where the refresh is warranted, but performance is still of utmost consideration, individual fields can be designated for refresh rather than the entire object. Within the "RuleWrite Information" dialog, with "Enable full refresh of all bound objects" unchecked, one can explicitly list the fields that are to be refreshed when the method is called. Checking the "Enable Recursion" checkbox for an entity field will recursively refresh all of the entity's children.

These optimizations are also obtainable by decorating your classes and/or methods with "InRuleImport*" attributes.

The Refresh Fields Action is also available in rule authoring and can be used to selectively refresh field values to working memory.

4.11 InRule Culture Settings

The culture settings of InRule are saved with a baseline of default English (United States) settings. At runtime or during authoring, the culture-specific formatting for dates, datetimes, and numbers are translated into the Windows settings defined by the user session for the operating system. This is true throughout InRule for the rule authoring in irAuthor, rule testing in irVerify, and rule execution in irServer.

Generally speaking, a user can author rules in Europe and see the formats displayed that are specific to European users. When that rule application is saved, InRule translates relevant data to the assumed English (United States) default for storage. Opening the rule application in a different Windows setting will result in those relevant formats being translated from the English (United States) baseline into the culture settings of the current user. This allows for portability of the rules/entity model (rule application) and provides independence from conflicting user settings.

Some authoring considerations for Dates, DateTimes and Numbers are as follows:

- Multiple argument functions require quotes around European formatted numbers because comma is the separator for function arguments. For example, Round(number,digits) in European format must be entered as *Round('1000,987',2)*. The English (United States) default equivalent is *Round(1000.987,2)*.
- Data stored as date, datetime, or number types will display in culture specific settings in inline tables. Date/datetime/number values stored in a Text type column are not adjusted per culture setting.

Since the culture specific settings of InRule are adopted from the user settings by default, the culture formats can be overridden at the thread level using .NET class libraries. By overriding the culture settings of the thread, developers can serve the culture settings to InRule calls. Sample source code for overriding the thread culture to English-United Kingdom is as follows:

```
using System.Threading;
using System.Globalization;

...

    CultureInfo newCulture = new CultureInfo("en-GB");
    Thread.CurrentThread.CurrentCulture = newCulture;
    session.ApplyRules();
```

See also Overriding Culture Settings at Runtime

4.12 Interacting with Non .NET Platforms

Calling the Rule Engine from COM

A COM-compliant typelib can be provided with the InRule installation to fully integrate the rule engine with a COM application.

Once set as a reference, all objects, methods, and events exposed by the irSDK will become available to the application.

Calling the Rule Engine from Java or other languages

The best approach for interacting with the rule engine from a non-Microsoft programming language is by writing a custom web service.

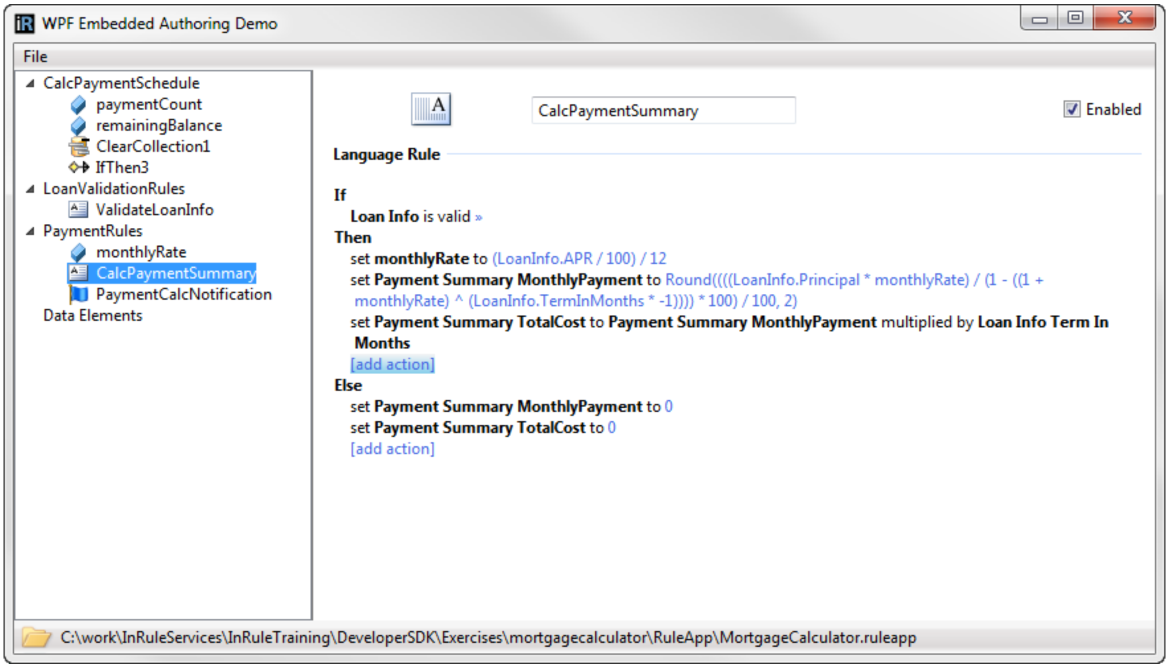
4.13 Embedding Authoring Controls

Almost all of the controls used to edit rules in irAuthor can be embeddable in custom .NET and WPF applications. This allows users to create or modify rule applications or parts of a rule application inside of their own custom .NET solution. This provides a domain-centric authoring experience with minimal coding efforts.

WPF controls can be embedded in WinForm application. Please see Embedding InRule Controls in WinForms for details.

The following is an example of a WPF application hosting the Business Language Editor. The tree on the left is a standard WPF tree which leverages the Authoring Framework Image Service to display the appropriate image for each rule definition.

Note: To load InRule resources, the Control Factory must be created prior to the loading of any windows.



CalcPaymentSummary

Language Rule

If
Loan Info is valid »
Then
set **monthlyRate** to (LoanInfo.APR / 100) / 12
set **Payment Summary MonthlyPayment** to Round((((LoanInfo.Principal * monthlyRate) / (1 - ((1 + monthlyRate) ^ (LoanInfo.TermInMonths * -1)))) * 100) / 100, 2)
set **Payment Summary TotalCost** to **Payment Summary MonthlyPayment** multiplied by **Loan Info Term In Months**
[add action]
Else
set **Payment Summary MonthlyPayment** to 0
set **Payment Summary TotalCost** to 0
[add action]

C:\work\InRuleServices\InRuleTraining\DeveloperSDK\Exercises\mortgagecalculator\RuleApp\MortgageCalculator.ruleapp

The following topics and examples explain how to embed InRule Controls

- The Process of Embedding InRule Controls
- WinForm Considerations

4.13.1 The Process of Embedding InRule Controls

Assemblies required
The minimal assemblies required for embedding InRule controls for WPF (unless otherwise noted) include the following:

Assembly Name
InRule.Authoring
InRule.Authoring.BusinessLanguage
InRule.Authoring.Editors
InRule.Authoring.Windows
InRule.Common
InRule.Repository

WPF assemblies are located in <InRule installation directory>\irSDK\bin.

Control Factory
The Control Factory is a class in The Rule Authoring Framework that abstracts much of the code required to create an instance of a control that can edit an InRule definition object (Entity, Rule Set, Rule, etc). The job of the Control Factory is to provide a control based on the type of object which is passed in.

Process

The typical process for embedding one of InRule's controls in WPF is as follows:

1. Create an instance of the ControlFactory
2. Load the Rule Application
3. Load the Rule Application into the Control Factory
4. Get the desired rule to be edited
5. Call the GetControl method on the Control Factory passing the definition object to be edited.
6. Load the control into the form. A common approach is to use the ContentControl available in WPF.

Sample code:

```
// Create control factory
var _controlFactory = new ControlFactory();

// Load the rule app
var _ruleAppDef = RuleApplicationDef.Load(@"C:\work\MortgageCalculator.ruleapp");

// Load the rule app into the control factory
_controlFactory.OpenRuleApplication(_ruleAppDef);
RuleSetDef langRule = _ruleAppDef.GetRuleSet("ValidateLoanInfo");

// Get the control and load it into the content control in the form
contentControl.Content = _controlFactory.GetControl(langRule);
```

Please see Embedding InRule Default Editors for more details.

Saving

Any changes that are made in an embedded control must be saved back to the rule application. Some changes are saved automatically back to the rule application. However, it is best to call the SaveValues method as shown below to ensure everything is saved correctly. Here is the code required to perform the save.

Note: This saves changes the RuleApplicationDef that is in memory, saving back to the file system or Catalog is a separate operation.

```
public void SaveRule(IValidatingEditor editor)
{
    if (editor != null)
    {
          editor.SaveValues();
    }
}
```

Controls with Display Options

Using the ControlFactory's GetControl method as described above will return the entire control as it is seen in irAuthor. This includes a label and image based on the type of control, a text box for updating the object's Name and a check box for enabling/disabling rules (if applicable). There are cases where it may not be desirable to have the entire editor displayed in an application. For this reason, the most commonly used controls can also be embedded without these extra items. The controls available in this manner are the Business Language, Decision Table and Condition editors. See Embedding the Language Rule Editor, Embedding the Decision Table Editor and Embedding the Condition Editor for samples of each.

4.13.2 WinForm Considerations

Embedding the InRule WPF controls in a WinForm application works the same is it does with WPF. This includes the initialization of the ControlFactory, loading the controls and saving changes. There are, however, a couple considerations to be aware of.

ElementHost control
In order to host a WPF control in a WinForm application, the ElementHost control must be used. This control was built for this exact purpose and is not specific to InRule.

```
// Get the business language editor and load into ElementHost control
elementHost.Child = _controlFactory.GetControl(langRule);
```

WPF Application Object
WPF controls that have popups require an instance of the WPF Application Object. In a WPF application, this is automatically created. This is not the case in a WinForm application. For this reason, this object must be created manually. The Application also must be set to remain open until it is shut down through code, otherwise it will shut automatically when the WinForm applications determines it is no longer needed. The following code will open the Application object in a WinForm and keep it open until it is closed. Shutting down the Application object when the WinForm closes is the recommended approach.

```
// Create appliication object
System.Windows.Application app = new System.Windows.Application();

// Set shutdown mode to explicit
app.ShutdownMode = ShutdownMode.OnExplicitShutdown;

// Create a delegate that will close the app object when the WinForm closes
Closed += delegate { app.Shutdown(); };
```

4.14 Customizing irAuthor with Extensions

irAuthor was built with the goal of being much more configurable by end users than previous versions of the product. This was achieved by making a more modular product that can optionally load the different pieces of functionality that are shipped with InRule. Custom functionality can also be created by the end user in the form of extensions which can be loaded into irAuthor alongside or in place of system extensions that ship with InRule. With extensions, irAuthor can be modified to look and work how you want.

Extensions can be created by any developer that has irSDK installed on their machine. An extension is simply a .NET solution that references InRule assemblies, allowing developers to write code against the new authoring framework and its underlying services. These services provide the developer with simplified access to accomplish most authoring related goals.

The authoring framework is comprised of a set of .NET assemblies that provide the functionality in irAuthor. System extensions that provide the out of the box functionality in irAuthor are built with the framework, which is public so it can be used by non-InRule developers as well. Modifications could include adding, removing or modifying functionality and user interface components.

Extension examples to customize irAuthor could include:

Override default editors in irAuthor

- Create a custom version of the Set Value control to allow users to select from a dropdown list of domain specific values

Automate processes inside of irAuthor

- Launch irVerify in the context of a certain entity with a given test file without having to navigate away from the currently selected element
- Create rule elements with a defined pattern to simplify the creation of similar entities or rule sets
- Add pre-defined attributes to a rule element by toggling a button in the ribbon as opposed to have to type them in

Create custom views based on a user or role

- Add/Remove Navigation Bar items
- Add/Remove buttons from the Ribbon
- Create new windows in irAuthor to capture additional information

The following topics and examples explain how to create custom irAuthor extensions

- The Rule Authoring Framework
- Example of Creating an irAuthor Extension
- Extension Advanced Topics

4.14.1 The Rule Authoring Framework

The rule authoring framework contains a set of application services that are used by irAuthor to perform all authoring related tasks. These services are public and can be used in custom built extensions as well. Below is a list of all of the services, broken out by namespace, along with a description of the functionality each one provides.

Note: The rule authoring framework services are not web services, but instead a set of classes available to the developer to gain access to and work with the different parts of irAuthor and the loaded rule application.

InRule.Authoring.Services

SelectionManager	This service allows developers to get and set the currently selected item (the object currently being edited) and raise events when the selected item changes, allowing subscribers to respond as needed.
ContentManager	The Content Manager simply maps .NET objects to their registered editors. These objects are typically rule application definition objects, however, this service can be used for custom (non-InRule) defined objects and editors as well.
RuleApplicationService	Provides primary means of interacting with the rule application. Rule applications can be loaded, created, closed and modified using this service. Events can also be raised to notify subscribers of rule application changes.
ImageService	The Image Service provides small and large images for InRule objects; for example, when an Entity is passed to the service, it returns the entity image in the size specified. The image factory provides named runtime access to the over 300 images shipped with irAuthor.
ClipboardService	Provides clipboard operations for InRule objects.
CommandService	Manages command providers, which provide commands based on a given object.
InlineTableImportService	Manages inline table importers.
LoggingService	Handles centralized logging.
NavigationHistoryS	Manages navigation history.

ervice	
OptionsService	Manages application options.
SettingsStorageService	Storage for settings in the Isolated Storage store.

InRule.Authoring.Windows.Services

irAuthorShell	The Shell is the primary means of interaction for modifying the irAuthor user interface. This includes activities such as adding/ removing tabs, groups and buttons from the Ribbon, adding/ removing panes from the Navigation Bar and launching custom Property Pages or Tool Windows.
RecentlyUsedService	Manages the list of recently used rule applications.
SearchService	Manages saved searches.
TestService	Manages rule application testing.

InRule.Authoring.Core.Navigation.Services

BookmarksService	Manages the bookmarks

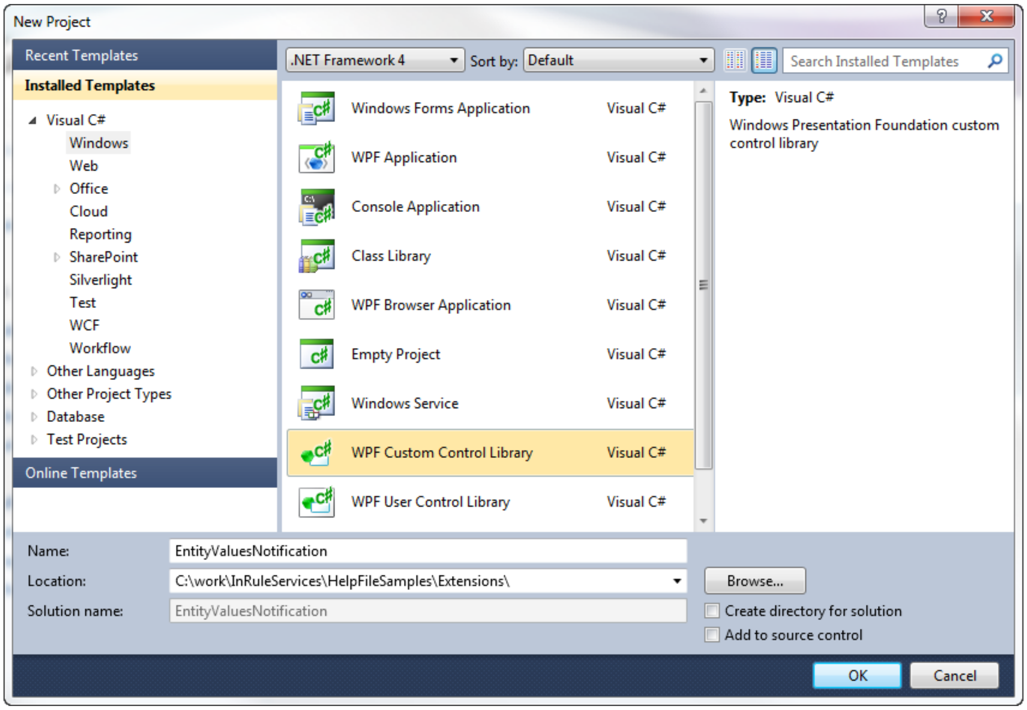
4.14.2 Example of Creating an irAuthor Extension

The following example walks through the process of creating an extension for irAuthor that will demonstrate:

- Adding a button to the Ribbon and using a Command to run user defined code
- Enabling and disabling of a button using authoring framework events
- Working with the Selection Manager Service
- Modifying the rule application through user defined code so irAuthor is aware of the changes using the Rule Application Service

Setting up a new Project

1. Create a new WpfCustomControlLibrary



Type: Visual C#

Windows Presentation Foundation custom control library

Name: EntityValuesNotification

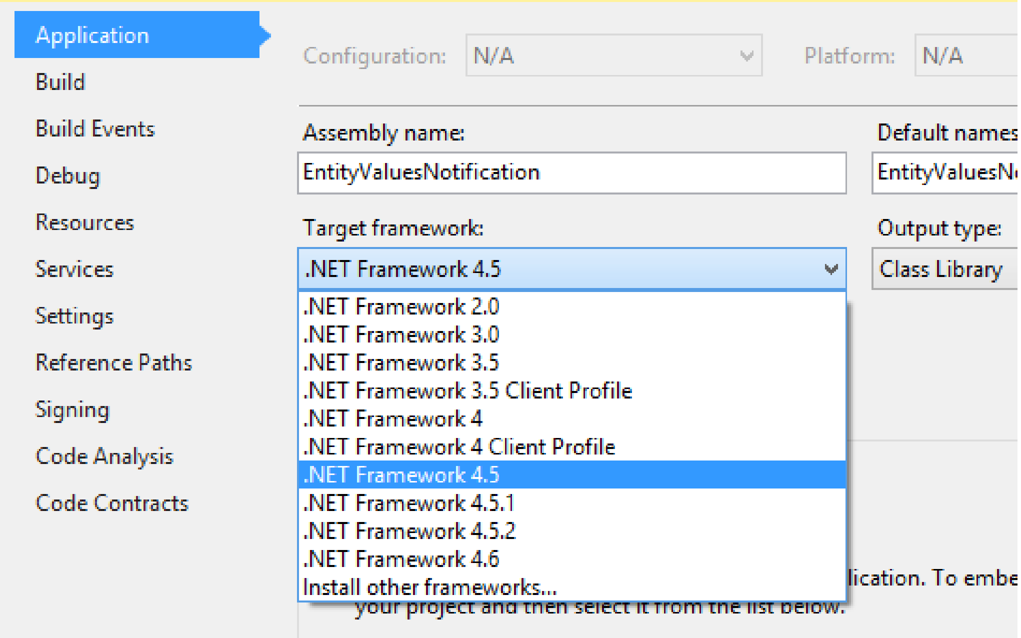
Location: C:\work\InRuleServices\HelpFileSamples\Extensions\

Solution name: EntityValuesNotification

□ Create directory for solution

□ Add to source control

2. Select the .NET Framework 4.7.2 option for the Target Profile on the Application property page.



Configuration: N/A

Platform: N/A

Assembly name: EntityValuesNotification

Default names EntityValuesN

Target framework: .NET Framework 4.5

Output type: Class Library

3. Delete the default CustomControl1.cs page
4. Delete the Themes folder
5. Add a new class called Extension.cs
6. Make the class Public
7. Add the following references (assemblies available from the InRule installation folder\irSDK

\bin)

```
Inrule.authoring.dll
Inrule.authoring.windows.dll
Inrule.common.dll
Inrule.respository.dll
```

Creating the extension class

1. Set the Extension class to inherit from the InRule.Authoring.Windows.ExtensionBase class

```
public class Extension:ExtensionBase
```

2. Create a default Constructor that will call the base class constructor with the name, description and Guid for the extension. The built in Guid generator in Visual Studio can be used to create your Guid.

```
public Extension()
       : base("EntityViewer",
            "Generates a notification that dumps all fields and their
values into a notification",
            new Guid("{CD02ADCB-BC8A-4393-A8EA-2903D5A2AD11}"))
        {
        }
```

3. Create a method in the Extension class called Enable which is an abstract inherited member and must be implemented. This is the method that will be called when the extension is loaded and where the code will go that adds the button to the Ribbon and wires up events that will enable and disable our button when not applicable.

```
public override void Enable()
       {
       }
```

4. Add a private class variable of type VisualDelegateCommand which will be the command that executes when the button is clicked

```
private VisualDelegateCommand _notificationCommand;
```

5. In the Enable method, add a new group to the Ribbon passing in the text for the group and the image that will be displayed if there is not enough room in the Ribbon. The image below is retrieved using the ImageFactory, which contains all of the images that ship with InRule, each of which is available in this manner.

```
var group = IrAuthorShell.HomeTab.AddGroup("Notifications",
ImageFactory.GetImageAuthoringAssembly("/Images/
FireNotificationInfo16.png"));
```

6. Instantiate the notification command. The delegate command is passed the method (which will run when the button is clicked), button text, small button image, large button image and whether the button is enabled by default.

   Note: The AddEntityNotification method creation will be shown below.

```
_notificationCommand = new
VisualDelegateCommand(AddEntityViewerNotification, "Add entity viewer",
    ImageFactory.GetImageAuthoringAssembly("/Images/
FireNotificationInfo16.png"),
    ImageFactory.GetImageAuthoringAssembly("/Images/
FireNotificationInfo32.png"),
    false);
```

7. Add the notification command to the group.

   Note: Several buttons could be added to a single group. Buttons can be added or removed to existing groups as well.

   ```
   group.AddButton(_notificationCommand);
   ```

8. The button is now added, next wire up the events that will enable/disable it. In this case, the button should only be enabled when the selected item is a rule set and when a rule application is opened. To wire up the events, we use the RuleApplicationService and SelectionManager as shown below.

   ```
   RuleApplicationService.Opened += SetEnabled;
   RuleApplicationService.Closed += SetEnabled;
   SelectionManager.SelectedItemChanged += SetEnabled;
   ```

9. The SetEnabled method will set the IsEnabled property on the command which will enable/disable the button.

   ```
   private void SetEnabled(object sender, EventArgs eventArgs)
   {
       // make sure we are on a rule element
       var selectedDef = SelectionManager.SelectedItem as RuleElementDef;
       if (selectedDef != null)
       {
           // if the selected def has an entity and rule set, this is a
   entity based rule or ruleset
           _notificationCommand.IsEnabled = ((selectedDef.ThisEntity !=
   null) && (selectedDef.ThisRuleSet != null));
       }
       else
       {
           // if this is not a rule element, always disable it
           _notificationCommand.IsEnabled = false;
       }
   }
   ```

10. Finally, create the method that will use the InRule SDK to create the notification. Most of the method below is standard SDK code, which has not changed in the new release of InRule. The actual adding of the notification to the RuleSet does use the new Rule Application service so the rest of the application knows that it was added, such as the tree.

    ```
    private void AddEntityViewerNotification(object obj)
    {
        // get selected rule set
        var ruleElementDef = SelectionManager.SelectedItem as RuleElementDef;

        if (ruleElementDef != null)
        {
            // get the ruleSet, should not be null or button would be
    disabled
            var ruleSetDef = ruleElementDef.ThisRuleSet;
    ```

```
        // get the entity that the ruleset is authored in (assumes entity
based rule set)
        var entityDef = ruleSetDef.ThisEntity;

        var s = new StringBuilder();
        s.AppendFormat("{0} field values:{1}", entityDef.Name,
Environment.NewLine);

        // spin through entity fields and generate the notification
        foreach (FieldDef fieldDef in entityDef.Fields)
        {
            s.AppendFormat("{0}: <% {0} %> {1}", fieldDef.Name,
Environment.NewLine);
        }

        // create notification and set message text
        var notificationDef = new FireNotificationActionDef();
        notificationDef.NotificationMessageText = s.ToString();

        // add notification to the rule set using the Controller
        RuleApplicationService.Controller.AddDef(notificationDef,
ruleSetDef);
    }
}
```

Deploying the Extension
After compiling the project, to deploy it, simply copy the DLL to the Extensions folder underneath the location where the irAuthor.exe is running. This is typically located at <InRule installation directory>\irAuthor\Extensions. irAuthor must be closed to copy the DLL if it already existed in the Extensions folder. After copying, launch irAuthor and go to File -->Extensions to enable the extension.

Note: You will need to close and re-open irAuthor in order to see a newly added extension.

4.14.3 Extension Advanced Topics

Creating System Extensions
It is possible to create an extension that cannot be disabled by the user without physically deleting the DLL file. This approach was taken due to the fact that users typically do not have permissions to delete files under Program Files. To create a system extension, when calling the constructor of the BaseExtension class, use the overload which accepts an isSystemExtension parameter. Simply pass in a value of true to make a system extension.

The following code is the same as the previous example, except true is being passed in for the final parameter (isSystemExtension), which makes it a system extension.

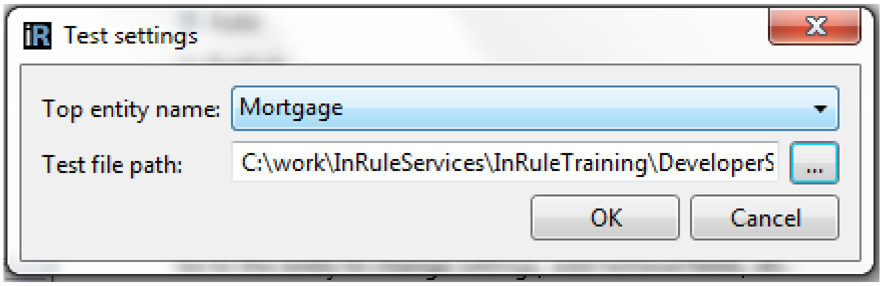
```
public Extension()
    : base("EntityViewer",
    "Generates a notification that dumps all field values into a notification",

    new Guid("{CD02ADCB-BC8A-4393-A8EA-2903D5A2AD11}"),
    true)
{
}
```

The Window Factory
The WindowFactory is a static class that can be used to create windows inside of irAuthor to capture information. Windows are made by creating a WPF User Control. Xaml can be used to built out entire forms with all of the standard WPF controls. The WindowFactory can then be leveraged to load

the forms from inside the extension.

Here is an example of a custom window that was created in an extension:



Top entity name: Mortgage

Test file path: C:\work\InRuleServices\InRuleTraining\DeveloperS

Assuming a WPF User Control is created and named TestDataControl, the following code will create an instance of the window and launch it inside of irAuthor.

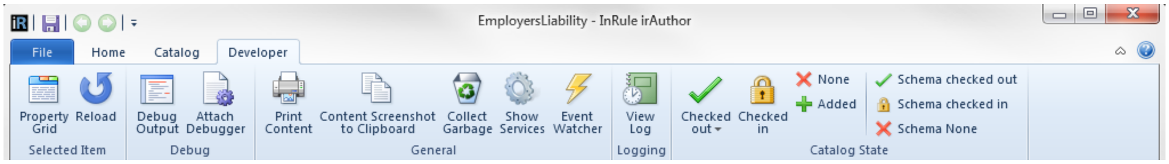
```
// Create an instance of the user control
var control = new TestDataControl(settings);

// Use the window factory to create the window passing the description, control
and required buttons
var window = WindowFactory.CreateWindow("Test settings", control, "OK",
"Cancel");

// Subscribe to the click event to run the desired code
window.ButtonClicked += myClickEvent;
window.Show();
```

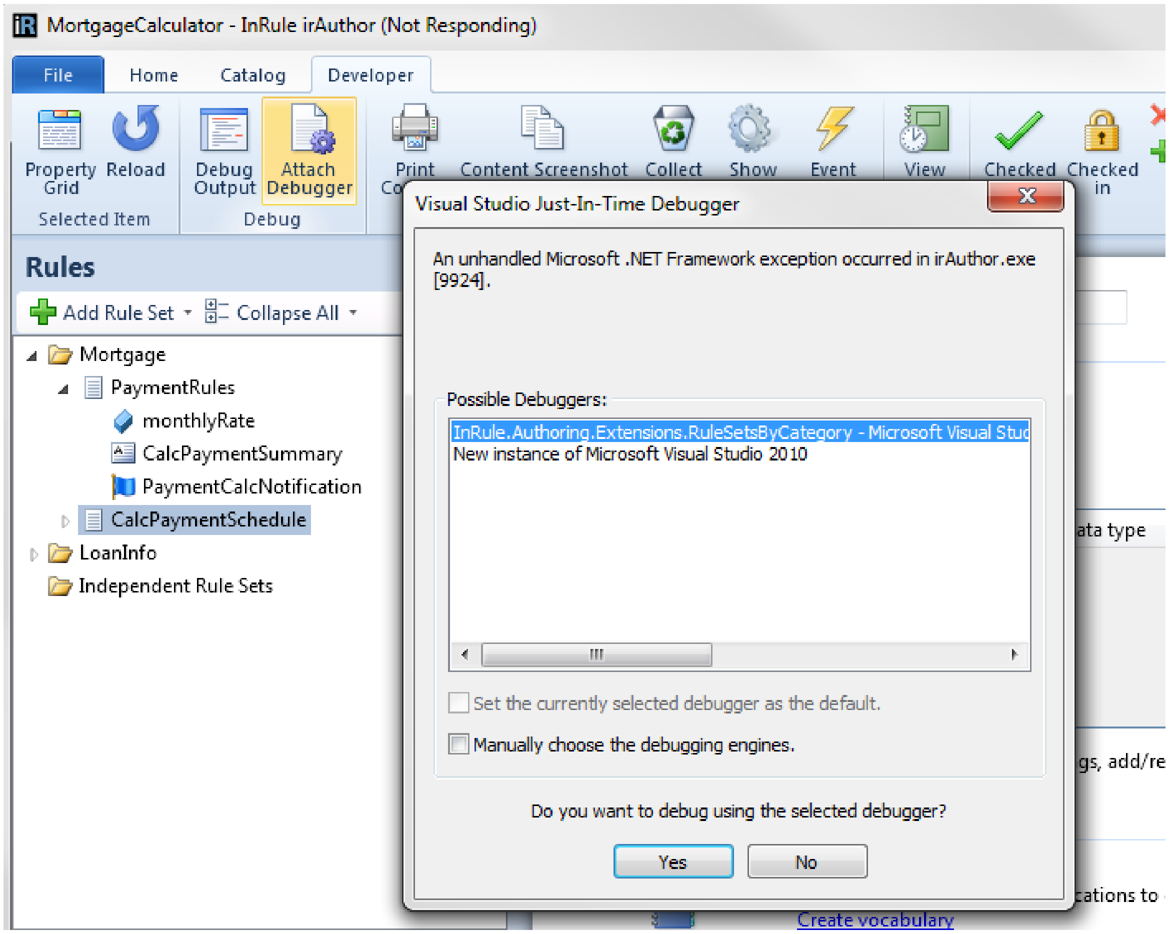
Debugging with Developer extension
InRule ships with a Developer extension that is very useful when creating an irAuthor Extension. The extension is not enabled by default, but can be enabled by going to File --> Extensions. Check the check box next to "Developer" to enable the extension.

Once enabled, a new tab in the Ribbon labeled "Developer" will appear, as shown here:



Below is a list of utilities available along with a description of each.

Attach Debugger
This is by far the most useful utility available. It allows an extension developer to go into irAuthor and attach to the instance of their Visual Studio project that contains the extension code. Once attached, all of the developers code is available for debugging. This means you can set a break point in the code, click the button created by the extension and step through the code. This greatly increases the ability to create and debug irAuthor extensions.



Visual Studio Just-In-Time Debugger

An unhandled Microsoft .NET Framework exception occurred in irAuthor.exe [9924].

Possible Debuggers:

InRule.Authoring.Extensions.RuleSetsByCategory - Microsoft Visual Stud
New instance of Microsoft Visual Studio 2010

□ Set the currently selected debugger as the default.

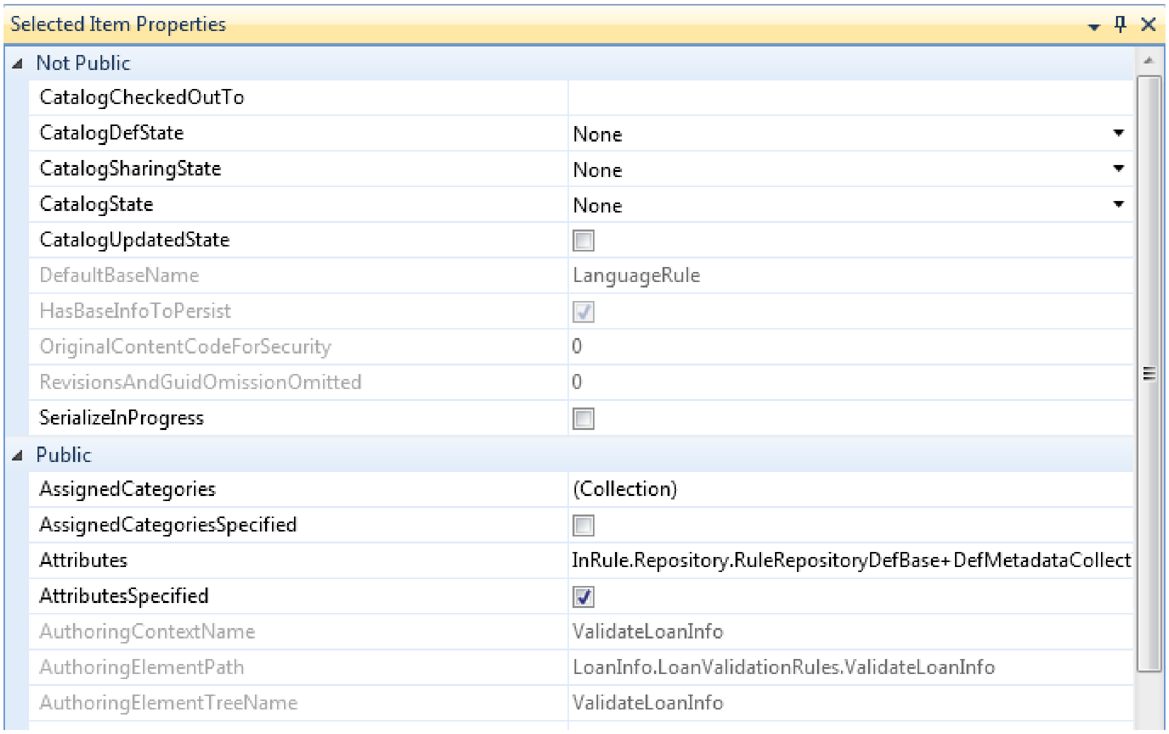
□ Manually choose the debugging engines.

Do you want to debug using the selected debugger?

The process is as follows:

1. Click the Attach Debugger button – This will launch the standard Visual Studio attachment window
2. Select the instance of Visual Studio that has the desired extension project open
3. Set a break point in the solution
4. Go back to irAuthor and perform the action (button click, select item, etc.) that will trigger the extension code
5. The break point should be hit and you can step through the code

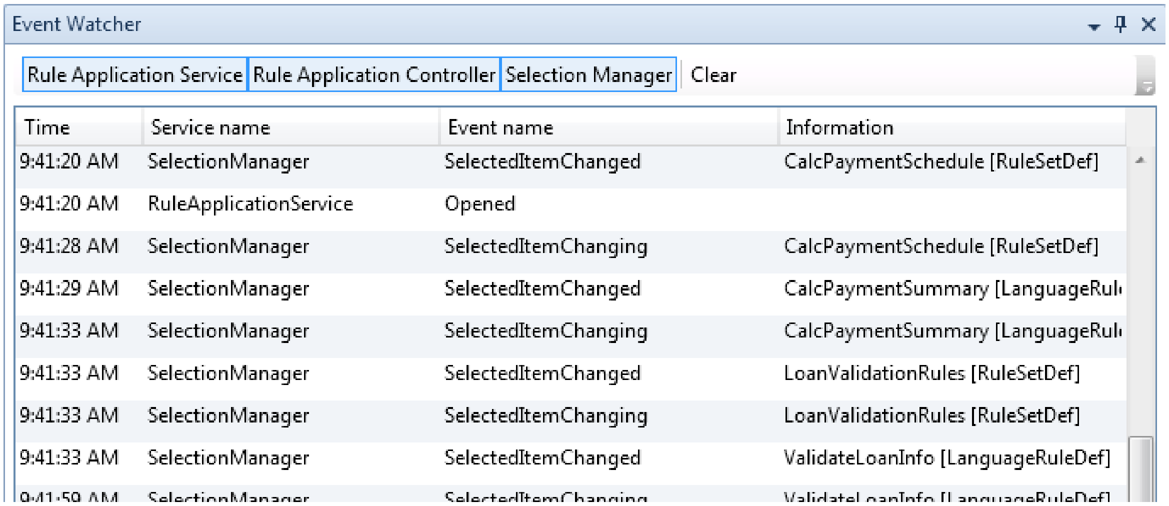
Property Grid
The Property Grid displays all public and private properties for the currently selected object. This is useful for scenarios where a developer wants to create their own View and needs to see which properties are available for binding to the View Model.



Selected Item Properties

Not Public	
CatalogCheckedOutTo	
CatalogDefState	None
CatalogSharingState	None
CatalogState	None
CatalogUpdatedState	□
DefaultBaseName	LanguageRule
HasBaseInfoToPersist	☑
OriginalContentCodeForSecurity	0
RevisionsAndGuidOmissionOmitted	0
SerializeInProgress	□
Public	
AssignedCategories	(Collection)
AssignedCategoriesSpecified	□
Attributes	InRule.Repository.RuleRepositoryDefBase+DefMetadataCollect
AttributesSpecified	☑
AuthoringContextName	ValidateLoanInfo
AuthoringElementPath	LoanInfo.LoanValidationRules.ValidateLoanInfo
AuthoringElementTreeName	ValidateLoanInfo

Event Watcher
The event watcher will display relevant events that are firing in irAuthor. This assists a developer when they are trying to figure out which events to attach to for their extensions.



Time	Service name	Event name	Information
9:41:20 AM	SelectionManager	SelectedItemChanged	CalcPaymentSchedule [RuleSetDef]
9:41:20 AM	RuleApplicationService	Opened	
9:41:28 AM	SelectionManager	SelectedItemChanging	CalcPaymentSchedule [RuleSetDef]
9:41:29 AM	SelectionManager	SelectedItemChanged	CalcPaymentSummary [LanguageRul
9:41:33 AM	SelectionManager	SelectedItemChanging	CalcPaymentSummary [LanguageRul
9:41:33 AM	SelectionManager	SelectedItemChanged	LoanValidationRules [RuleSetDef]
9:41:33 AM	SelectionManager	SelectedItemChanging	LoanValidationRules [RuleSetDef]
9:41:33 AM	SelectionManager	SelectedItemChanged	ValidateLoanInfo [LanguageRuleDef]
9:41:59 AM	SelectionManager	SelectedItemChanging	ValidateLoanInfo [LanguageRuleDef]

Using post build events to simplify testing
To simplify testing when creating an extension, make a post-build event that will copy the compiled DLL to the Extensions folder where irAuthor is running. irAuthor will need to be closed each time so the file copies successfully.

Example:
copy $(TargetPath) <InRule installation directory>\irAuthor\Extensions\

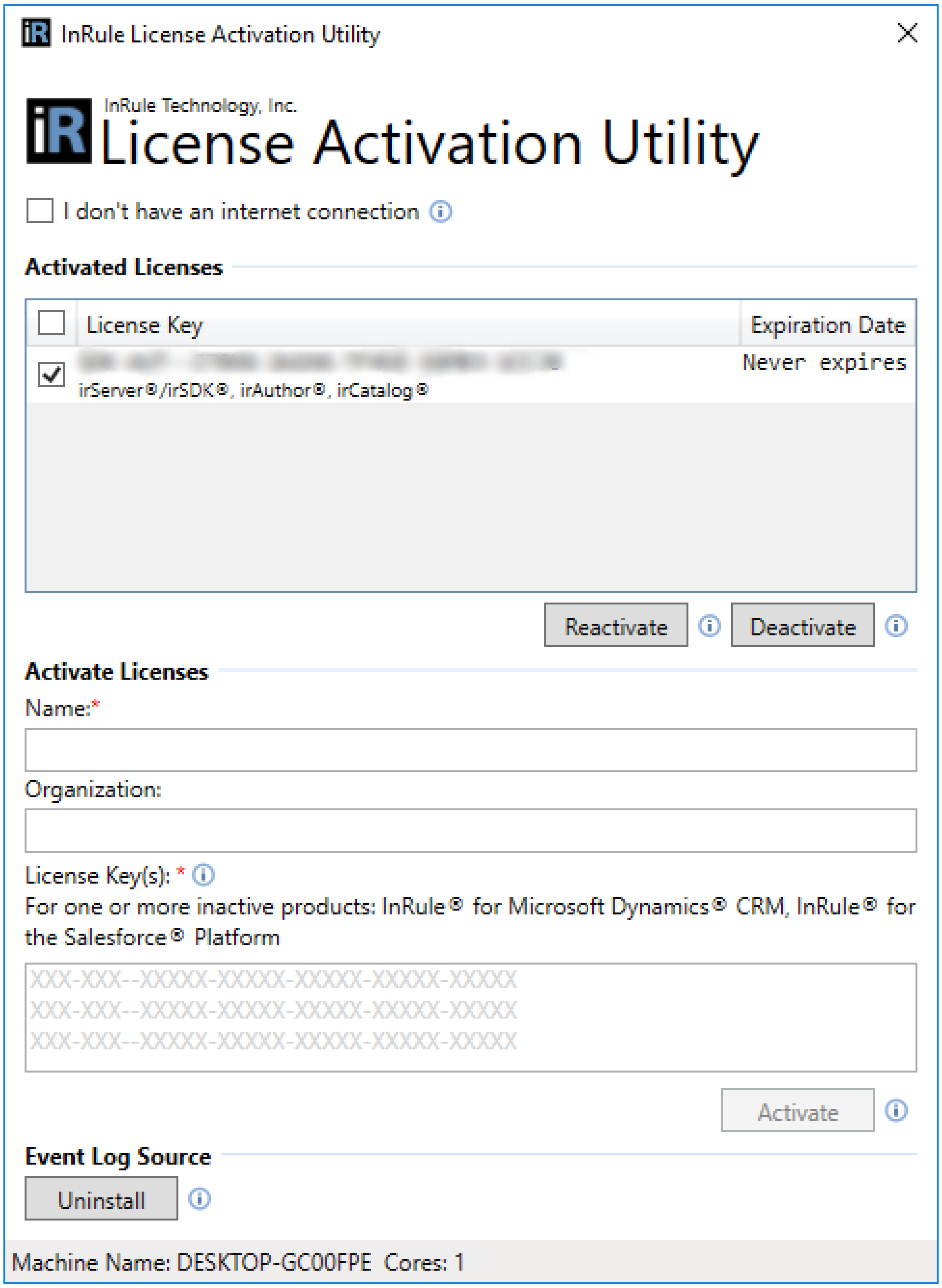
4.15 License Activation Utility

The License Activation Utility for XCOPY deployments

InRule supports a License Activation Utility that allows a user to activate and deactivate licenses and/or register the Event Log Source without running the MSI installer. The Activation Utility is a WPF application with a full Windows UI, but it will run silently if started with command line arguments. The InRule License Activation Utility ships with versions 4.1.3 and later.

After installation of any InRule component, the ActivationUtility.exe file will be available in the <InRule installation directory>\ActivationUtility folder. This folder can be XCOPY deployed to machines that require deployment, license activation and/or Event Log Source registration without running the InRule MSI installer.

A sample screen shot of the utility is below. A given machine only requires activation one time, and then InRule DLLs may be deployed to any location on that machine.



InRule License Activation Utility

InRule Technology, Inc.

License Activation Utility

□ I don't have an internet connection

Activated Licenses

□ License Key	Expiration Date
☑ [redacted] irServer®/irSDK®, irAuthor®, irCatalog®	Never expires

Activate Licenses

Name:*

Organization:

License Key(s): *

For one or more inactive products: InRule® for Microsoft Dynamics® CRM, InRule® for the Salesforce® Platform

XXX-XXX--XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
XXX-XXX--XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
XXX-XXX--XXXXX-XXXXX-XXXXX-XXXXX-XXXXX

Event Log Source

Machine Name: DESKTOP-GC00FPE Cores: 1

Activating a new license

When you receive a license key for an InRule product or feature, you need to activate it before it can be used on the machine. To do this:

- If your computer is not connected to the internet or otherwise cannot access the InRule license server, check the "I don't have an internet connection" box near the top of the form.
- In the "Activate Licenses" section, enter your name and organization.
- Enter your license key in the space provided. You may enter more than one key, one per line.
- Click Activate.

Reactivating an existing license

If your time-bound license has expired or is about to expire, you can reactivate or refresh it once InRule updates the license server with your new expiration date.

- If your computer is not connected to the internet or otherwise cannot access the InRule license server, check the "I don't have an internet connection" box near the top of the form.
- In the "Activated Licenses" section select the affected key(s).
- Click Reactivate.

Deactivating an existing license

You may also deactivate a license in order to free it up for use on another computer.

- If your computer is not connected to the internet or otherwise cannot access the InRule license server, check the "I don't have an internet connection" box near the top of the form.
- In the "Activated Licenses" section select the key(s) you wish to deactivate.
- Click Deactivate.

Managing the event log

You may install or uninstall the InRule Event Log source. For more information, see Event Log Details.

Note: Unlike the InRule Installer, the Activation Utility does not automatically attempt to create an InRule-specific Event Log Source/section. In order to perform this recommended function, you will need to run the utility by right-clicking and selecting "Run as administrator". Installation and un-installation of the InRule Event Log Source is then accomplished using the button at the bottom of the Activation utility's GUI. The action is independent of activating/deactivating licenses but can be performed in the same session.

Command Line interface

In addition to the Windows user interface, the License Activation Utility also supports a command line interface so that it can be invoked from scripts or code. If a given operation is successful, the process will exit a code of zero. If the operation fails, then a non-zero exit code will be returned. During execution, all output is written to the Windows Standard Output Stream as well as a log file.

The following command line arguments are supported:

-u <username> (required)	Sets the name of the user that will be logged with the license activation
-o <organization>	Sets the name of the organization that will be logged with the license activation
-a <license key>	Activates InRule components for a specific license key over an available Internet connection
-d	Deactivates all found InRule components over an available Internet connection
-dl <license key>	Deactivates all found InRule components for a specific license key over an available Internet connection
-cs	Checks the status of InRule components. License status messages are written to

	the output stream.
-il	Installs InRule Event Log Source
-ul	Uninstalls InRule Event Log Source
-h or -?	Display Help

Activation Utility command-line code samples (using C#):

The following code sample demonstrates using the command line to activate a license key.

```
var exePath = "<some path to ActivationUtility.exe>";
var commandLineArgs = "-u someusername -o someorgname -a xxx-xxx--xxxxx-xxxxx-
xxxxx-xxxxx-xxxxx";
using (var process = new Process())
{
    process.StartInfo.FileName = exePath;
    process.StartInfo.RedirectStandardOutput = true;
    process.StartInfo.UseShellExecute = false;
    process.StartInfo.Arguments = commandLineArgs;
    process.Start();
    // It is important that the following call is made before the
process.WaitForExit.
    // Otherwise a deadlock can occur.
    var commandOut = process.StandardOutput.ReadToEnd();
    process.WaitForExit();
    var exitCode = process.ExitCode;
    Console.WriteLine("Installation process exit code: {0}", process.ExitCode);
    Console.WriteLine(commandOut);
    process.Close();
}
```

The following code sample demonstrates using the command line to install the Event Log Source.

```
var exePath = "<some path to ActivationUtility.exe>";
var commandLineArgs = "-il";
using (var process = new Process())
{
    process.StartInfo.FileName = exePath;
    process.StartInfo.RedirectStandardOutput = true;
    process.StartInfo.UseShellExecute = false;
    process.StartInfo.Arguments = commandLineArgs;
    process.Start();
    // It is important that the following call is made before the
process.WaitForExit.
    // Otherwise a deadlock can occur.
    var commandOut = process.StandardOutput.ReadToEnd();
    process.WaitForExit();
    var exitCode = process.ExitCode;
    Console.WriteLine("Installation process exit code: {0}", process.ExitCode);
    Console.WriteLine(commandOut);
    process.Close();
}
```

Logging:

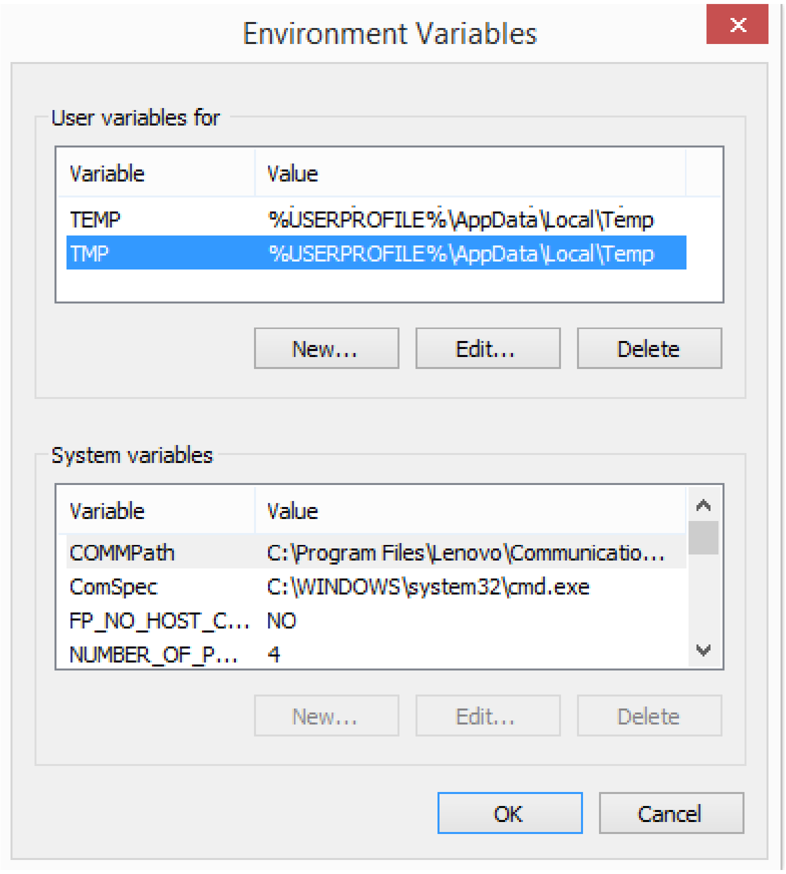
The Activation Utility logs to the %temp%\InRule\Logging\InRule.Licensing.Log file.

You can modify the log level and/or how and if this information is logged to the Event Log by modifying the Activation Utility's configuration file, ActivationUtility.exe.config, located in the same directory as the Activation Utility.
See InRule Logging Config File Settings for additional details.

4.16 InRule Temp Files

InRule will periodically read and write to the temp directory. Therefore the user account which the app domain is running under needs to have sufficient permissions to the temp directory, so that it can read/write files and directories.

The temp directory used will be whatever %TMP% is set to, which may depend on whether the account has user environmental variables:



Environment Variables

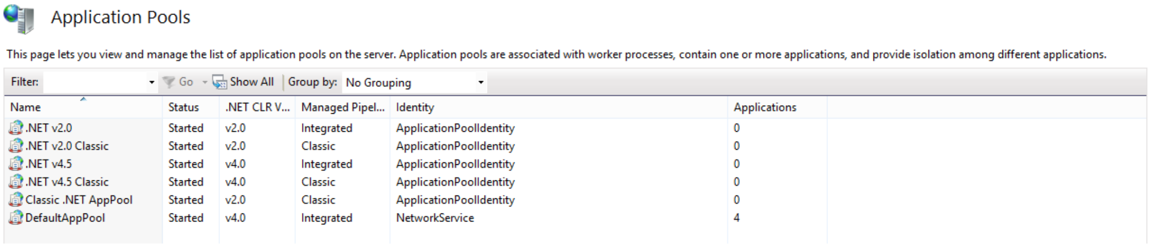
User variables for

Variable	Value
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
COMMPath	C:\Program Files\Lenovo\Communicatio...
ComSpec	C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	4

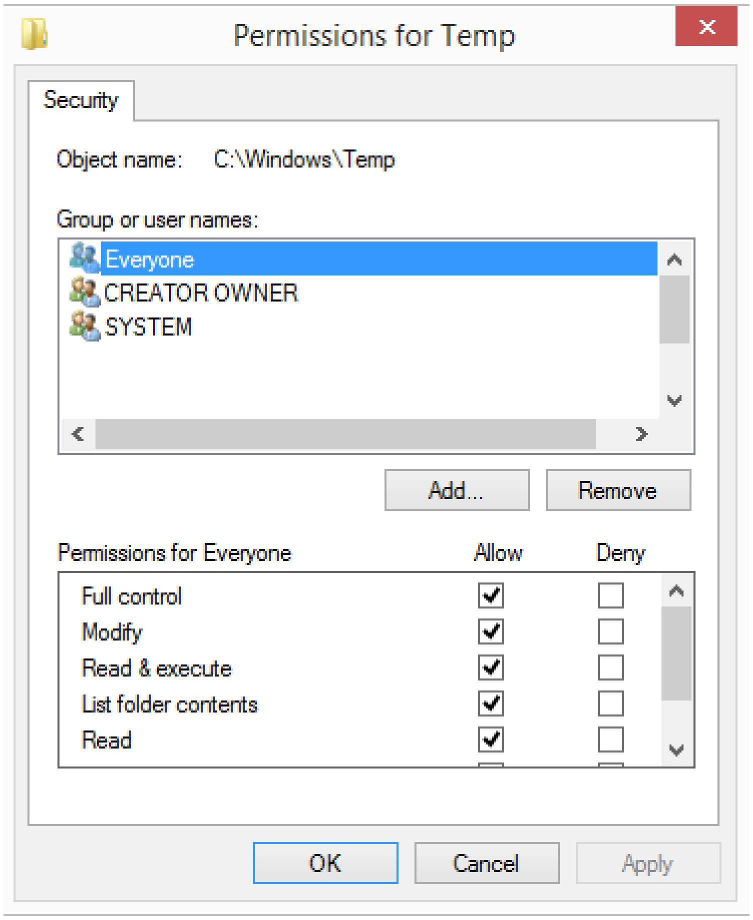
If you are running InRule from an application managed by IIS, then the user account will be the identity of the application pool. Therefore, you need to assure that the account has full access to the temp directory, or you can grant 'Everyone' full access, as shown below:



Application Pools

This page lets you view and manage the list of application pools on the server. Application pools are associated with worker processes, contain one or more applications, and provide isolation among different applications.

Name	Status	.NET CLR V...	Managed Pipel...	Identity	Applications
.NET v2.0	Started	v2.0	Integrated	ApplicationPoolIdentity	0
.NET v2.0 Classic	Started	v2.0	Classic	ApplicationPoolIdentity	0
.NET v4.5	Started	v4.0	Integrated	ApplicationPoolIdentity	0
.NET v4.5 Classic	Started	v4.0	Classic	ApplicationPoolIdentity	0
Classic .NET AppPool	Started	v2.0	Classic	ApplicationPoolIdentity	0
DefaultAppPool	Started	v4.0	Integrated	NetworkService	4



Permissions for Temp

Object name: C:\Windows\Temp

Group or user names:

- Everyone
- CREATOR OWNER
- SYSTEM

Permissions for Everyone	Allow	Deny
Full control	☑	□
Modify	☑	□
Read & execute	☑	□
List folder contents	☑	□
Read	☑	□

4.17 irSDK for .NET Core

InRule support for .NET Core takes the form of additional support in the NuGet packages. The `InRule.Common`, `InRule.Repository`, and `InRule.Runtime` packages contain libraries that target both the full .NET Framework 4.7.2 and .NET Standard 2.0. This allows you to use the runtime SDK in either framework.

This topic describes this support in more detail.

Which Framework Should I Use?

The choice of whether to use the full .NET Framework or the .NET Core Framework is not a simple one. .NET Core is Microsoft's future direction, and their guidance is to use .NET Core, *unless you can't*.

InRule recommends .NET Core in these situations.

- You want to migrate away from the Windows platform
- You are beginning a new development project
- You want to take advantage of the newest framework technologies
- You expect a performance benefit from using .NET Core

Here are some reasons you might want to use the full .NET Framework.

- You have an existing implementation with a relatively small number of irServer cores
- You require functionality that .NET Core does not provide

You should not attempt to mix the full .NET Framework and the .NET Core Framework in the same application. Doing so would give unpredictable results.

Setting up irSDK for .NET Core

To use the irSDK support in .NET Core, first install InRule version 5.1.0 or later on a Windows computer, selecting the irSDK component. This places the NuGet packages in the install directory, under `irSDK\NuGetPackages`. From there, deploy the NuGet packages as you would otherwise.

Based on your license type, you may need an additional license to use the .NET Core Framework.

Of course, you still have to activate irSDK. On a non-Windows computer, the InRule Activation Utility is not available because it has a Windows user interface. Instead, you may download the .NET Core activation utility from the InRule support site, support.inrule.com. This utility is a command line program which you must run on the computer where you deploy irSDK.

To run the Activation Utility on Linux, for example, enter the command `dotnet ActivationUtility.dll`, followed by command-line parameters, as outlined in License Activation Utility. Since the Windows Event Log is not supported on non-Windows platforms, the parameters `-il` (Install Log) and `-ul` (Uninstall Log) are not supported.

After completing license activation on Linux, the `InRuleLicense.xml` file will be stored by default in the home directory of the user that activated the license, e.g. `~/.inrule/InRuleLicense.xml`. This file may be moved to `/etc/inrule/InRuleLicense.xml` so that irSDK may be licensed for the system, not just the current user. The license file may also be copied to the directory of the application that consumes irSDK.

irSDK Limitations in .NET Core

At this time, .NET Core support consists strictly of the irSDK runtime. These InRule components are not included:

- irServer Rule Execution Service. You may engage InRule ROAD Services if you want help writing a service wrapper.
- irAuthor, irCatalog, irWord, etc.

There is no installation program, per se.

Some Rule Actions are not available in .NET Core.

- Execute Web Service (Execute REST Service is supported)
- Execute Workflow
- Execute SQL Query with an Oracle or OLE DB Endpoint (SQL Server and SQLite Databases will work as expected)

irAuthor: irAuthor does not currently identify unsupported Rule Actions.

The .NET Assembly Schema element supports a .NET Standard assembly, but not a .NET Core assembly.

POSIX: irSDK for .NET Core does not support en-us-POSIX.

InRule Catalog Service: The InRule Catalog Service uses `WsHttpBindings` by default, but .NET Core can only use `BasicHttpBindings`.

Here is a sample WCF configuration for the IIS Catalog Service:

```
<system.serviceModel>
  <bindings>
    <wsHttpBinding>
      <binding name="WSHttpBinding" maxReceivedMessageSize="50000000">
        <readerQuotas maxStringContentLength="50000000" />
        <security mode="None">
          <!-- **WARNING** Changes to the security binding must also be made in
               client binding code -->
          <transport clientCredentialType="None" proxyCredentialType="None" />
          <message clientCredentialType="None" />
        </security>
      </binding>
    </wsHttpBinding>
    <basicHttpBinding>
      <binding name="BasicHttpBinding" maxReceivedMessageSize="50000000">
        <readerQuotas maxStringContentLength="50000000" />
          <security mode="None">
            <transport clientCredentialType="None" proxyCredentialType="None" />
          </security>
      </binding>
    </basicHttpBinding>
  </bindings>
  <services>
    <service name="InRule.Repository.Service.RepositoryService"
             behaviorConfiguration="repositoryServiceBehavior">
      <endpoint address="" binding="wsHttpBinding"
                bindingConfiguration="WSHttpBinding"
                contract="InRule.Repository.Service.ICatalogServiceContract" />
      <endpoint address="core" binding="basicHttpBinding"
                bindingConfiguration="BasicHttpBinding"
                contract="InRule.Repository.Service.ICatalogServiceContract" />
      <!-- ** NOTE Metadata not supported SSL at this time **-->
      <!--<endpoint address="mex" binding="mexHttpBinding"
                    contract="IMetadataExchange"/>-->
    </service>
  </services>
  <behaviors>
    <serviceBehaviors>
      <behavior name="repositoryServiceBehavior">
        <serviceDebug httpHelpPageEnabled="true" />
        <!-- Enables the IMetadataExchange endpoint in services that -->
        <!-- use "metadataSupport" in their behaviorConfiguration attribute. -->
        <!-- In addition, the httpGetEnabled and httpGetUrl attributes publish -->
        <!-- Service metadata for retrieval by HTTP/GET at the address -->
        <!-- "http://localhost:8080/SampleService?wsdl" -->
        <!-- ** NOTE Metadata not supported SSL at this time **-->
        <!-- <serviceMetadata httpGetEnabled="true" httpGetUrl=""/>-->
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <serviceHostingEnvironment multipleSiteBindingsEnabled="True" />
</system.serviceModel>
```

Here is a sample WCF configuration for Windows Service Catalog Service:

```
<system.serviceModel>
  <bindings>
    <wsHttpBinding>
      <binding name="WSHttpBinding" maxReceivedMessageSize="50000000">
        <readerQuotas maxStringContentLength="50000000"
maxNameTableCharCount="50000000" />
        <security mode="None">
          <!-- **WARNING** Changes to the security binding must also be made in
               client binding code -->
          <transport clientCredentialType="None" proxyCredentialType="None" />
          <message clientCredentialType="None" />
        </security>
      </binding>
    </wsHttpBinding>
    <basicHttpBinding>
      <binding name="BasicHttpBinding" maxReceivedMessageSize="50000000">
        <readerQuotas maxStringContentLength="50000000" />
          <security mode="None">
            <transport clientCredentialType="None" proxyCredentialType="None" />
          </security>
      </binding>
    </basicHttpBinding>
  </bindings>
 <services>
   <service name="InRule.Repository.Service.RepositoryService"
            behaviorConfiguration="repositoryServiceBehavior">
     <endpoint address="http://computername:8082/InRuleRepositoryService_v5.0.29"
               binding="wsHttpBinding" bindingConfiguration="WSHttpBinding"
               contract="InRule.Repository.Service.ICatalogServiceContract" />
     <endpoint address="http://computername:8082/InRuleRepositoryService_v5.0.29/
core"
               binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding"
               contract="InRule.Repository.Service.ICatalogServiceContract" />
     <!--<endpoint address="mex" binding="mexHttpBinding"
                   contract="IMetadataExchange" />-->
   </service>
 </services>  <behaviors>
    <serviceBehaviors>
      <behavior name="repositoryServiceBehavior">
        <serviceDebug httpHelpPageEnabled="true" />
        <!-- Enables the IMetadataExchange endpoint in services that -->
        <!-- use "metadataSupport" in their behaviorConfiguration attribute. -->
        <!-- In addition, the httpGetEnabled and httpGetUrl attributes publish -->
        <!-- Service metadata for retrieval by HTTP/GET at the address -->
        <!-- "http://localhost:8080/SampleService?wsdl" -->
        <!-- ** NOTE Metadata not supported SSL at this time **-->
        <!-- <serviceMetadata httpGetEnabled="true" httpGetUrl=""/>-->
      </behavior>
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
```

Known Issues

At runtime, Windows and Linux behave differently in some cases.

Culture-specific string comparisons: On Windows, comparing `"Straße" = "Strasse"` results in true. On Linux, the same comparison results in false.

Date round tripping: Converting a date to a string and back can give incorrect results. For example, 1/16/2067 converts to the string "1/16/67" which then converts to the date 1/16/1967.

InRule recommends you do not convert dates to strings.

Serialization with BinaryFormatter: When serializing Entity or Session state and using a bound .NET Assembly configured to use BinaryFormatter, any DateTime.Kind will not be preserved when deserializing; it will always default to DateTimeKind.Unspecified. This is a known issue in .NET Core.

The RuleApplicationDef or any Def instance that includes a collection will fail to serialize with BinaryFormatter because the .NET Core version of CollectionBase is not marked as `[Serializable]`. This affects both Windows and Linux.

Logging

InRule uses the Windows Event Log for its logging. Since this is not available on other operating systems, any attempt to use it will fail in .NET Core.

Performance Considerations

In benchmark tests at InRule, the same Rule Application was run on various operating systems and platforms. Here are the execution times, in milliseconds.

Platform	Windows	Linux
.NET Framework 4.6.2	1423	N/A
.NET Core 2.0.7	1382	2097
.NET Core 2.1.0 Preview 2	1312	1555
Mono 5.10	1628	1898

This shows that the .NET Core Framework generally outperforms the full .NET Framework, and that it is improving. It also shows that .NET Core on Windows performs better than the same test on Linux.

Part V

Implementation Guide

5 Implementation Guide

Overview

This guide is intended to provide high-level architectural and design guidance regarding the implementation of InRule® within single applications and across disparate systems.

Before embarking on any InRule implementation, the following concepts should be considered:

- How many different business problems will the rule implementation need to solve? Do components and services need to be reusable? How many different rule applications are required?
- What is the anticipated load the rule implementation needs to support?
- Do the applications and business problems require stateful execution, or can a stateless architecture be used?
- Which technology stacks will be included in the integration with rules?
- What types of system architectures are best suited for scalability, portability, and integration with disparate systems?
- As the solution is developed and deployed, which configuration settings and SDK approaches will optimize maintenance and performance?
- How will rule changes be managed across development, testing, and production environments?

This section includes five major sub-sections that assist in answering the questions posed above:

- High-level Implementation Guidelines
- Application Architectures
- Common Integration Scenarios
- Rule Deployment
- Performance Tuning and Best Practices

5.1 High-level Implementation Guidelines

Overview

This section contains information on general architectural concepts that are imperative to any successful rule integration project.

- Anatomy of a Rule Request
- Typical Rule Request Workflow
- Service-Oriented Rule Architectures

5.1.1 Anatomy of a Rule Request

Integration to Execute Rules at Runtime

Before rules can be consumed at runtime by production applications, some integration points must be defined between calling applications and the rule engine. In general, at least a small amount of .NET code must be written to create these integration points.

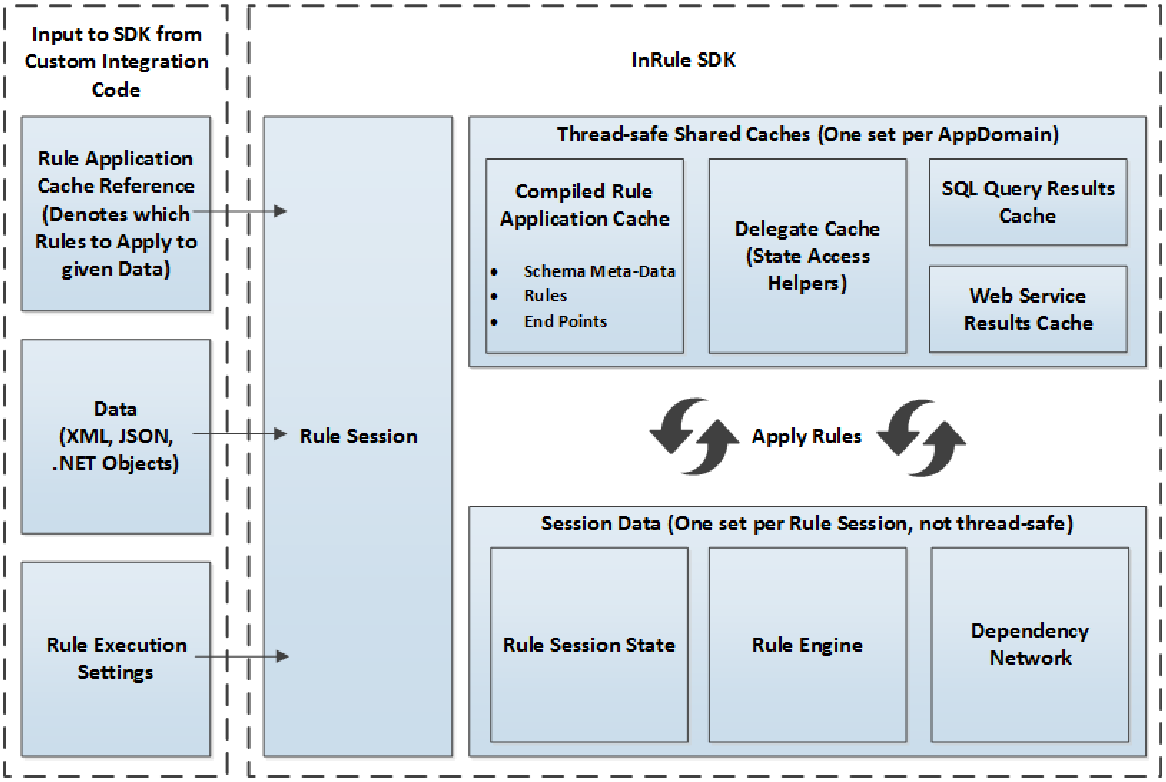
Due to the fact that the InRule SDK is written in .NET code, there are many possible application architectures where similar and simple integration code and approaches can be reused. Essentially most places that a .NET application can run are also possible places where InRule can run. The InRule integration can normally be accomplished with just a few lines of .NET code, and then exposed to any number of consuming applications using reusable DLL or service interfaces.

Runtime rule execution is accomplished by creating a "rule request" to the rule engine, and then consuming output as necessary after the rule engine has completed executing rules. A rule request is minimally comprised of the following components that must be instantiated by application integration code:

- **Rule Session -** the RuleSession is a public class in the InRule SDK. It serves as the main programming interface for developers that are invoking the rule engine. The Rule Session automatically manages communication between user-provided data, rules, and rule engine operations.
- **Entity (Application Data) -** At least one InRule Entity must be created by the integration code. The Entity is a wrapper that the rule engine uses to read and write to data provided by calling applications. The underlying objects that are wrapped by the Entity instances are generally custom .NET types, XML, or JSON. However, InRule also supports the ability to construct custom data providers if other backing stores are required. Root Entities must be created explicitly in integration code, but children of these root Entities will be automatically generated by the rule engine during rule processing.
- **Rule Application Reference -** The Rule Session must be directed to run a specific set of rules against the Entity data that is provided. The Rule Session accepts a "rule application reference" that it can use to either automatically locate an existing rule application cache entry, or load rules and populate the rule application cache on-demand.
- **Rule Execution Settings (Optional) -** Prior to execution, the developer can adjust rule execution behavior using an extensive list of settings exposed by the InRule SDK. In many cases the default settings are adequate for a given production application; in these cases no settings need to be provided by the developer.

After a Rule Session has been populated with the required input, rules can be invoked by issuing either an Apply Rules or Execute Rule Set call to the Rule Session.

The Anatomy of a Rule Request


Input to SDK from Custom Integration Code
Rule Application Cache Reference (Denotes which Rules to Apply to given Data)
Data (XML, JSON, .NET Objects)
Rule Execution Settings
InRule SDK
Rule Session
Thread-safe Shared Caches (One set per AppDomain)
Compiled Rule Application Cache
Schema Meta-Data
Rules
End Points
Delegate Cache (State Access Helpers)
SQL Query Results Cache
Web Service Results Cache
Apply Rules
Session Data (One set per Rule Session, not thread-safe)
Rule Session State
Rule Engine
Dependency Network

Free Threaded Caching per .NET App Domain

The Rule Session automatically manages four thread-safe caches that are maintained by the InRule SDK. Most developers do not need to write any code against these caches, but they should be aware that they exist and affect performance and memory usage of the rule engine.

- **Compiled Rule Application Cache -** This cache contains the compiled rule applications that have been consumed by the given .NET AppDomain. A compiled rule application contains related sets of rules, along with dependent end point and schema definitions for those rules. When a rule request is sent to the engine, rule applications are compiled on demand and added to this cache if they are not already present
- **Delegate Cache -** This cache contains state access delegates that are used by the rule engine to read and write to state. This cache is populated on demand during rule execution.
- **SQL Query Results Cache -** This cache contains record sets returned from Execute SQL Query rules and Value Lists. This cache is populated on-demand, and is not used if no rules integrate with SQL queries.
- **Web Service Results Cache** - This cache contains result sets returned from rules that execute web service calls. This cache is populated on-demand, and is not used if no rules integrate with web services.

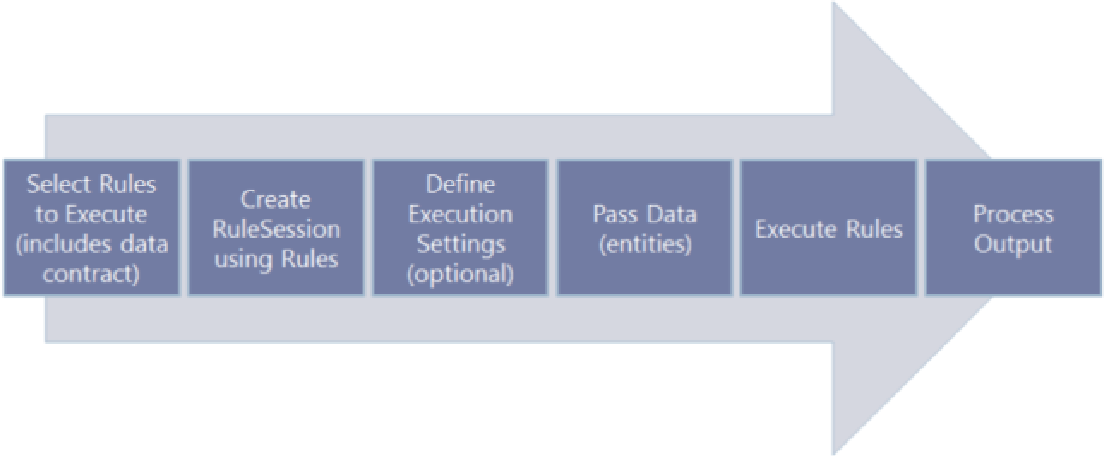
5.1.2 Typical Rule Request Workflow

The Life Cycle of a Typical Rule Execution Request

The main steps of a typical rule execution request in an application are as follows:

1. Create a Rule Application Reference
2. Create a Rule Session and pass it the Rule Application Reference

3. Set rule execution settings or end point overrides for the Rule Session
4. Create an Entity which will be used to pass data (state) to the rule engine
5. Execute Rules against the data
6. Work with output


Select Rules to Execute (includes data contract)
Create RuleSession using Rules
Define Execution Settings (optional)
Pass Data (entities)
Execute Rules
Process Output

The code for a sample rule execution is included below:

```
public string RunRules(string xmlInput)
{
    // Create rule application reference
    var ruleApp = new FileSystemRuleApplicationReference(@"C:\RuleApps
    \MyRuleApplication.ruleappp");

    // Create a rule session
    using (var session = new RuleSession(ruleApp))
    {
        // Adjust settings or endpoint overrides if needed

        // Create at least one entity
        var entity = session.CreateEntity("MyEntity", xmlInput);

        // Execute auto rules
        session.ApplyRules();

        // Return output -- this example extracts XML
        return entity.GetXml();
    }
}
```

Step 1 - The Rule Application Reference

The Rule Application Reference is an object that points to a rule application. A rule application is a collection of rule sets along with the data schemas and end point definitions that support those rule sets. The reference contains enough meta-data so that rules can be loaded and compiled on demand, or reused from the Rule Application Cache if already previously compiled. The Rule Application Reference has three implementations which can be used to load rules from various sources:

- **CatalogRuleApplicationReference -** refers to a rule application stored in irCatalog®

- **FileSystemRuleApplicationReference -** refers to a rule application stored on a Windows file system
- **InMemoryRuleApplicationReference -** refers to a rule application read into memory from any source that can produce rule application XML, such as a content management system or database

Step 2 - The Rule Session

- The Rule Session is the main point of interaction between custom SDK integration code and the rule engine
- The Rule Session requires a Rule Application Reference that is defined in Step 1 above. The rule applications used by the reference are typically created with irAuthor®; however they can also be built using a custom authoring solution built against irSDK or the InRule embeddable authoring controls.
- Note that the RuleSession implements IDisposable, and should be explicitly or implicitly Disposed after it is no longer used. For stateless architectures, the .NET Using block is recommended. For stateful architectures, the RuleSession should be disposed when deemed no longer necessary by the given session management solution.

Step 3 - Execution Settings and End Point Overrides

- There are several execution settings that control cache, execution, and parallelism behaviors. In general the default settings should be used, but if necessary they can be overridden before executing any rules in the rule session
- If web service or database endpoints are used in rules, then the URLs and connection strings can be overridden for any given environment. This allows different services or databases to be used in different environments without the need to update rules. All end point overrides should be applied before executing rules.

Step 4 - Create an Entity

At least one "root" entity must be created that will allow the rule engine to communicate with data that is passed to the rules. The rule engine will automatically create child entities as necessary, so those entities do not need to be explicitly created.

InRule supports a variety of ways with which the rule application schema can be defined. The schema definition type chosen also dictates the ways with which the state can be passed into the engine. The schema for a rule application can be defined the following ways:

- Schema can be imported from one or more .NET assemblies. This schema model facilitates passing state to the rule engine via .NET object instances.
- Schema can be imported from an XSD. This schema model will result in the developer passing state as an XML documents or strings.
- Schema can also be defined ad-hoc in irAuthor. State is then defined using the RuleSession class to create Entities and set Field values or passing in XML or JSON.
- Schema can be imported from Microsoft® Dynamics® CRM if using the irX® authoring add-in. This schema model will be backed by XML during runtime.

Generally speaking, there are some relevant statements worth mentioning about schema definition and state models:

- Often, an external schema definition will already exist. Importing an external definition as the rule application schema will enforce consistency between the calling application and InRule. Rule applications with an imported schema require a manual 're-sync' each time the external definition is adjusted. This forces users of InRule to verify that introduced schema changes do not cause issues with existing rule logic.
- Building an ad-hoc schema can be an agile approach to working with a rule application. However it may cause additional effort to map state values between external state objects and InRule state. For example, if a rule author builds existing fields into a rule application

schema, the developer is often accountable for seeing that those extra fields are mapped to corresponding input/output values for the calling application.

- Importing the schema from a .NET assembly adds the ability to call instance methods in the classes from rules. Further, passing state to the engine is straightforward because the rule session will accept a .NET object graph. During rule execution, the rule engine reads values from the object graph using the class's Get property accessors. Likewise, the rule engine will write object graph values using the class's Set property accessors.
- Importing from either .NET assembly or XSD will allow entities and fields to be individually aliased. During the import setup, individual entities and fields can also be included or excluded from the import.

Step 5 - Execute Rules

Now that the Rule Session is populated, rules can now be executed against the data that was provided to the Rule Session.

Most rule applications contain groups of rules defined as a series of "rule sets". A rule set is simply a group of rules. The exact rule sets that are executed and the order in which rules are executed are dependent on the fire modes, run modes, and data context of the given rule sets that are contained within a rule application.

Fire Modes

Rule Sets can be set to one of two "fire modes":

- **Auto** - The rule set will be available to automatically fire at least once for each instance of the entity type to which the rule set is associated. Depending on the "run mode" and "activation" settings of the rule set, the rule set may fire more than once or not fire at all.
- **Explicit** - The rule set will never fire automatically. It will only fire if explicitly invoked by a call to the InRule SDK, or when invoked by another rule.

Run Modes

The firing behavior of auto rule sets can be controlled by setting the "run mode" of the auto rule set:

- **Optimized** - Rules within the rule set may not fire in authored order, but are instead fired in the fastest order as determined by the rule engine. Rules may fire more than one time if values that they previously consumed are modified by another rule.
- **Sequential** - Rules within the rule set fire in authored order. The entire rule set (sequence) may fire more than one time if values that were previously consumed by the rule set are modified by other rules.
- **Single Pass Sequential** - Rules within the rule set fire in authored order. The rule set will not re-fire based on data changes made by subsequent rules.

The following additional rule constructs are also supported by InRule. These rules are always set to fire automatically (fire mode "auto"), and they may re-fire if the values that they previously consumed are modified by another rule.

- Calculations
- Classifications
- Constraints

Rule Context

The number of instances of a given rule and the data against which it operates is based on the "context" of that rule. In InRule, rules can be authored in the context of an entity or a field, or with no data context. If a rule or rule set is authored within the context of an entity or field, an instance of that rule set or rule is created and evaluated for each instance of the entity or field that is present within the given rule session. If a rule has a data context, it is evaluated against the entity instance with which it is associated.

InRule also supports authoring rules with no entity or field context. These rule sets are called "independent rule sets". An independent rule set must have data passed to it when it is invoked, since it has no inherent data context with which to begin reading data fields.

Running Rules

Rule execution can be initiated using one of three methods in irSDK:

- **RuleSession.ApplyRules** - Executes all "auto" rules against entity instances that have been added to the rule session. If auto rule sets contain rules that invoke explicit rule sets, then those explicit rule sets will also fire.
- **RuleSession.ExecuteIndependentRuleSet** - Executes a specific independent rule set given input arguments passed in the call. Any auto rules that are applicable will also fire.
- **Entity.ExecuteRuleSet** - Executes a specific explicit rule set against a given entity instance. If the rule set accepts parameters, then parameters can be passed in the call. Any auto rules that are applicable will also fire.

Rule Execution Order and Frequency of Rule Execution

During an ApplyRules request, the engine fires all Optimized and Sequential rule sets before firing any Single Pass Sequential rule sets. The Optimized and Sequential rule sets fire in the order that they are discovered as the engine reads in root entities and related child entities. Single Pass Sequential rule sets are fired last in the order that they were discovered during the initial read of entities.

When an ExecuteIndependentRuleSet or ExecuteRuleSet call is invoked, all Optimized and Sequential rule sets fire first in the order that they are discovered, followed by the explicit rule set that has been requested in the call. Finally, any Single Pass Sequential rule sets are fired in the order that they were discovered during the initial read of entities.

Other types of auto rules, such as Calculations, Classifications, and Constraints, are fired on demand as they are consumed by other rules or rule engine state reads.

If a RuleSession is reused for more than one request, then all Single Pass Sequential rule sets will re-fire on each subsequent rule request that is submitted to the same RuleSession. The following other "auto" rule types will re-fire only if the rule engine dependency network determines that some data change should cause them to re-fire:

- Sequential Rule Sets
- Optimized Rule Sets
- Calculations
- Classifications
- Constraints

Advanced Control over Rule Firing using Activation Settings

An advanced technique to control the firing of auto rule sets during rule execution is to employ an Activate/Deactivate pattern. This pattern can rely on the Activate Rule Set and Deactivate Rule Set actions in irAuthor, or rule sets can be activated and deactivated with code using irSDK.

When a rule set is activated it is added to the processing agenda of the rule engine so it is available to fire. When it is deactivated, it is removed from the agenda so that it will not fire. The rule author can control which "auto" rules are available for consideration at any point during rule processing by toggling the activation settings of rule sets.

Notes on activating or deactivating rule sets:

- Rule sets can be assigned to one or more 'Categories'. InRule contains actions for Activate/Deactivate by Category so that a rule author can activate or deactivate many different rule sets simultaneously based on the categories that are assigned to the rule sets.
- Explicit Rule Sets (which include Independent Rule Sets) also have properties for activation/deactivation. However, the rule compiler will not accept explicit rule sets as targets of activation or deactivation rules.

- Rule sets also contain a property for 'Enabled' which should not be confused as having similar functionality as 'Activated'. The 'Enabled' property cannot be controlled via irSDK or via irAuthor during rule execution. If a rule or rule set is not Enabled, it will not be included in the compiled rule application and will not be accessible in any way at runtime.

Step 6 - Consume Rule Engine Output

Once rule execution is complete the data that was passed into the rule engine is available for use elsewhere in the application. In the case of object instances, the actual object graph will be kept in sync during execution by the rule engine. XML or JSON data must be explicitly retrieved from the session.

In addition to the updated state data, the following are some of the more commonly used mechanisms to facilitate communication from the rule engine to the host application:

- **Notifications** - Messages generated by the Fire Notification action
- **Validations** - Error messages associated with a specific field or entity that are fired by the Mark Field Invalid action or constraints
- **Rule Execution Log** - The log contains execution information such as when there are state changes, which rules were fired and other runtime details
- **Trace** - If tracing was turned on in code, the trace file can be saved for review in irAuthor after execution completes. ***Please note that tracing is resource intensive and can significantly slow rule execution -- sometimes by more than a factor of ten. It is recommended that rule tracing is enabled only as needed, and not for every rule execution request. It is not recommended to use tracing for production applications.***

5.1.3 Service-Oriented Rule Architectures

Using InRule with Service-Oriented Architectures (SOA)

Due to the flexibility of the InRule SDK and the .NET framework, InRule can be configured to run in a vast array of possible application architectures. However, many customers today consider business rule processing to be an enterprise service, with the same service being available to many disparate systems. A single rule processing service may be required to process rules for many different business problems against many different data schemas. A platform-agnostic service interface and expose rule execution to many other platforms built on various technology stacks.

With InRule, it is possible to easily create a "generic" service interface using WCF or Web APIs that can accept and return many different data schemas. In addition, this service can accept input that denotes which rules should be run against a given set of data. A service with this interface is then capable of reuse across many different business problems and potential calling applications.

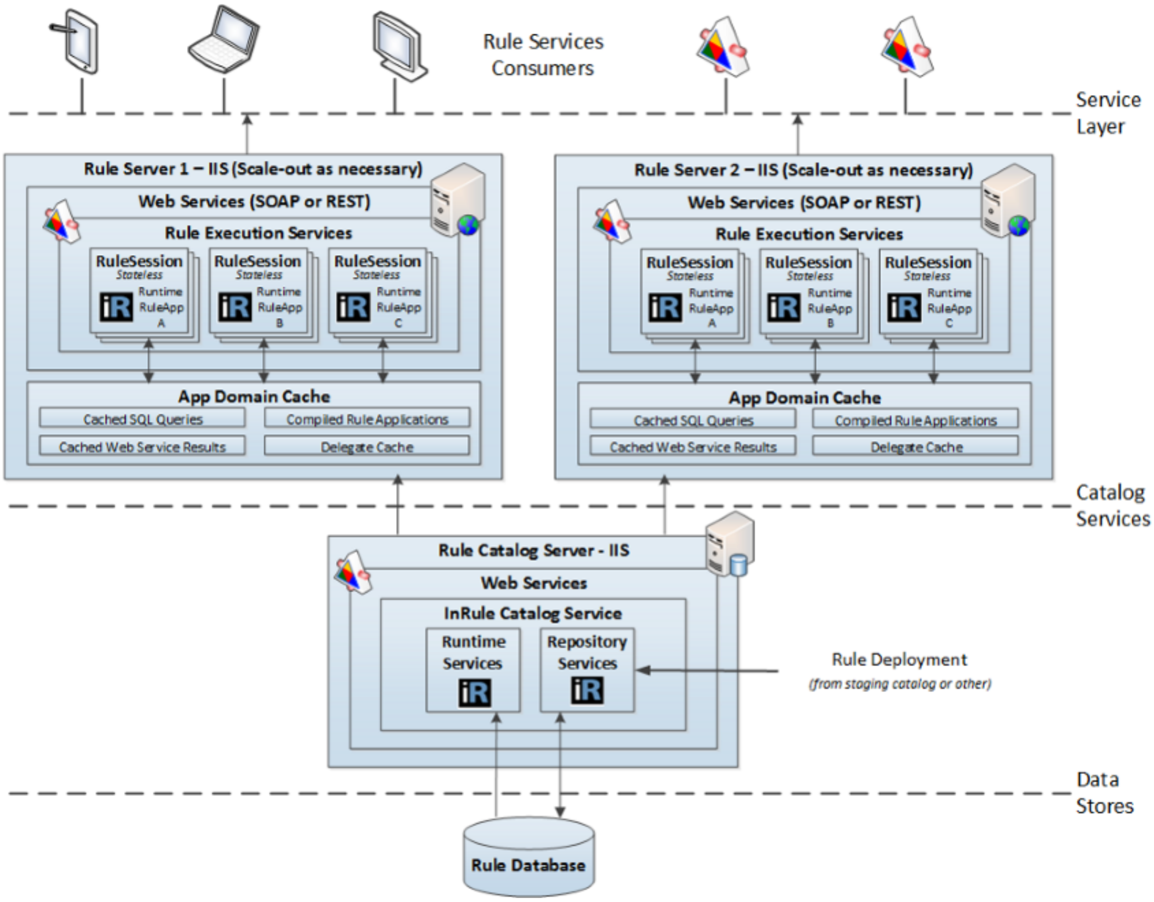
Most reusable services have some basic requirements to be considered "enterprise" services, all of which are accommodated by the WCF or Web API services approach and the free-threaded capabilities offered by InRule:

- High availability
- Scale-out easily as demand grows, without changing code
- Multi-threaded request handling

Note that InRule does offer stateful rule processing as part of the rule session, but most rule services are designed to be stateless so that each rule request is completely independent of every other rule request. In the stateless scenario, a rule session is only used for a single request and then discarded. The stateless architecture offers far more flexibility for scale-out and declarative deployment, and is therefore recommended over stateful approaches wherever possible.

A sample SOA for Business Rule processing.

Stateless rule service requests are spread across one or more identical rule processing servers.


Rule Services Consumers
Service Layer
Rule Server 1 – IIS (Scale-out as necessary)
Web Services (SOAP or REST)
Rule Execution Services
RuleSession Stateless Runtime RuleApp A
RuleSession Stateless Runtime RuleApp B
RuleSession Stateless Runtime RuleApp C
App Domain Cache
Cached SQL Queries
Compiled Rule Applications
Cached Web Service Results
Delegate Cache
Rule Server 2 – IIS (Scale-out as necessary)
Web Services (SOAP or REST)
Rule Execution Services
RuleSession Stateless Runtime RuleApp A
RuleSession Stateless Runtime RuleApp B
RuleSession Stateless Runtime RuleApp C
App Domain Cache
Cached SQL Queries
Compiled Rule Applications
Cached Web Service Results
Delegate Cache
Catalog Services
Rule Catalog Server - IIS
Web Services
InRule Catalog Service
Runtime Services
Repository Services
Rule Deployment
(from staging catalog or other)
Data Stores
Rule Database

5.2 Application Architectures

Overview

This section contains examples of common architectural patterns that are used with InRule implementations. Please be aware that many other configurations are possible.

- Rule Services - The most typical approach used to integrate InRule. A custom developed web service exposes rule execution to multiple consuming applications and user interfaces
- Web Applications - Typical web application with InRule running locally on each web server in a farm
- ESB and Messaging - Using InRule as an endpoint for an Enterprise Service Bus, where many disparate systems may reuse a rule service by passing messages to the service
- Job Scheduling and Batch Processing - Running rules against large volumes of data
- Desktop and Disconnected Applications - Running rules locally on desktops, laptops, or Windows tablets, even when disconnected from the enterprise network

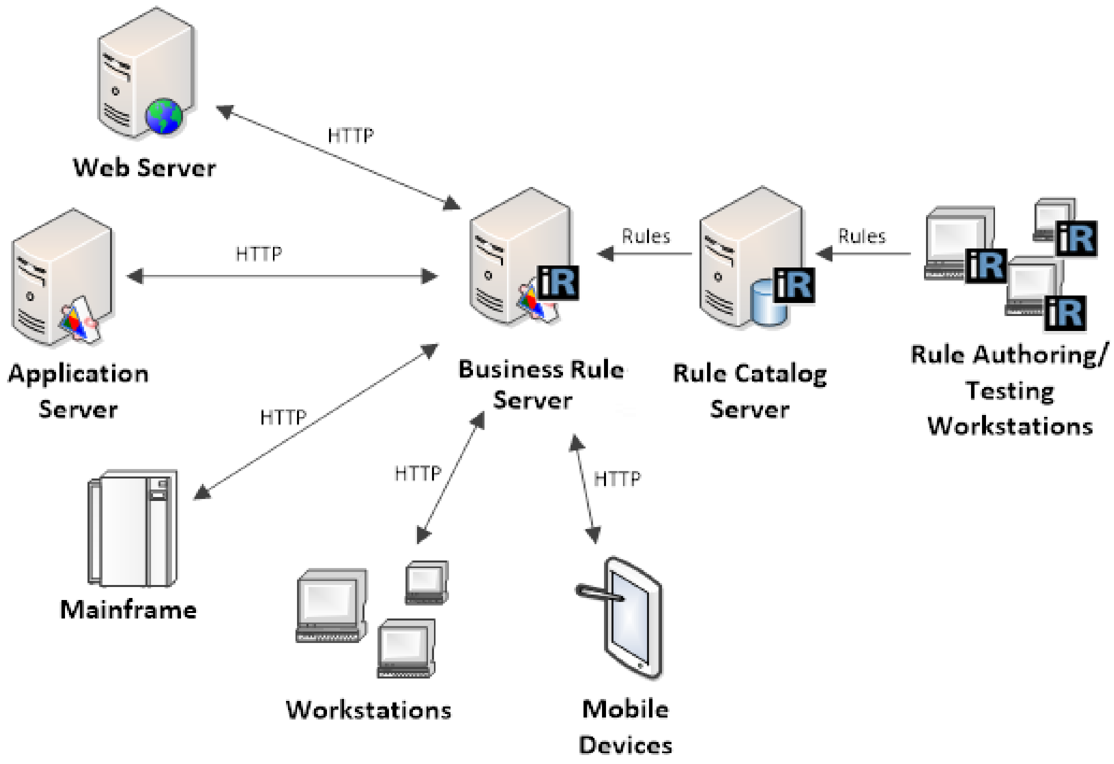
5.2.1 Rule Services

Running Rules from a reusable service layer

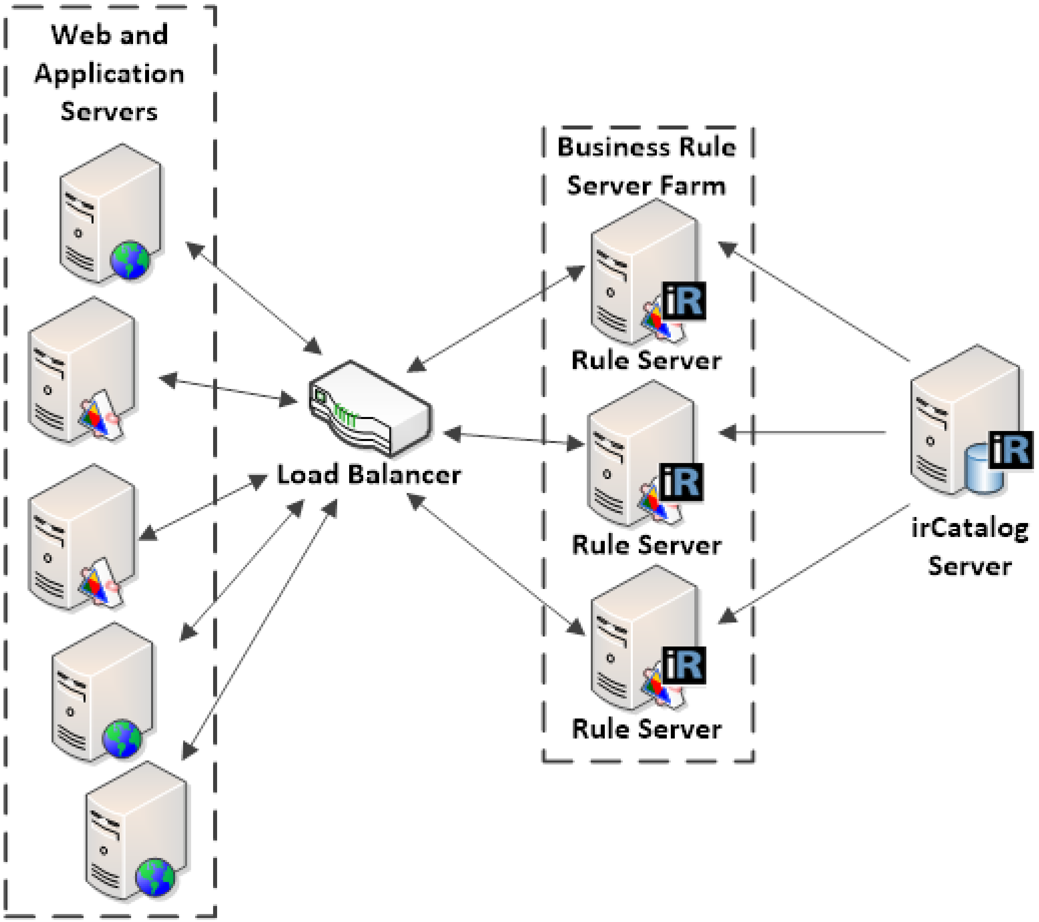
Key Points

- Rule execution occurs only on the Business Rule Server(s)
- Rule execution can scale-out with demand by spreading load across a server farm
- A single, generic, rule-processing web service interface can be reused for many different business problems
- Many different consuming applications and platforms can reuse the same rule-processing web service interface
- State is passed in the request and the response usually contains modified state and/or related messaging
- Rule engine instances are created using irSDK code within the .NET web service
- Business Rule Servers pull a fresh copy of the rules only as needed from the irCatalog Server
- Multiple Business Rule Servers can use a single irCatalog Server
- For smaller-scale implementations, the Catalog can be hosted on a Business Rule Server
- In scale-out and high-availability scenarios, a load balancer can be used to spread requests across a farm of servers

Running Rules with a Service-Oriented Architecture



Scaling-out the Service-Oriented Architecture to meet growing demand

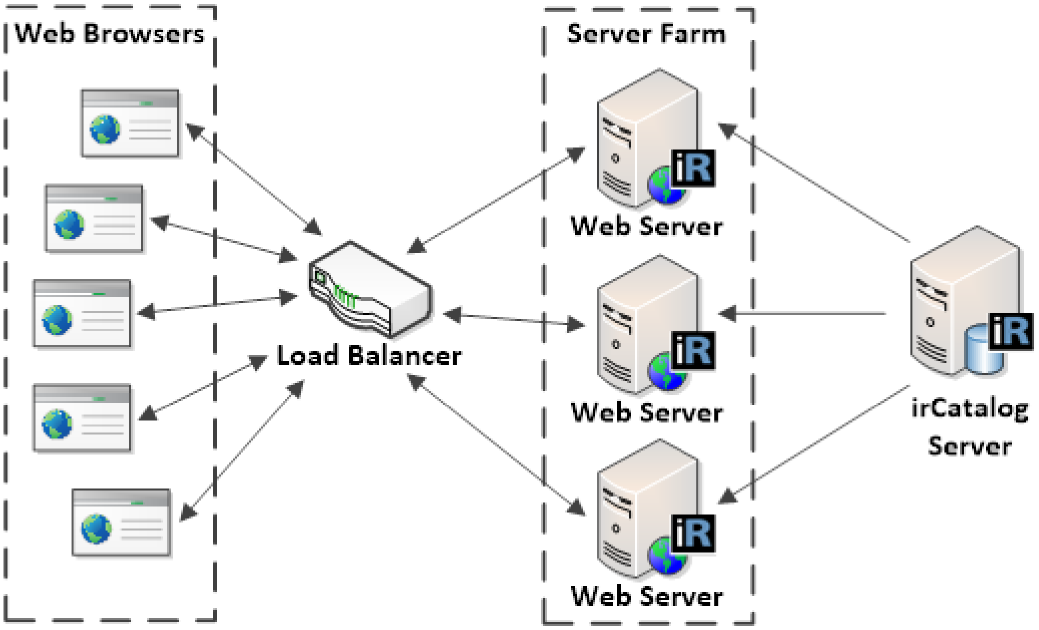


5.2.2 Web Applications

Running InRule from a Web Application In-Process

Key Points

- Rule Execution occurs on the web server(s)
- The rule engine is instantiated using irSDK code within the .NET web application
- Web servers can be load-balanced in a server farm
- Each web server pulls a fresh copy of the rules from irCatalog only as needed via the irCatalog Service
- A single irCatalog server can be used to support multiple web servers
- irCatalog may also be hosted on the same server as the web application

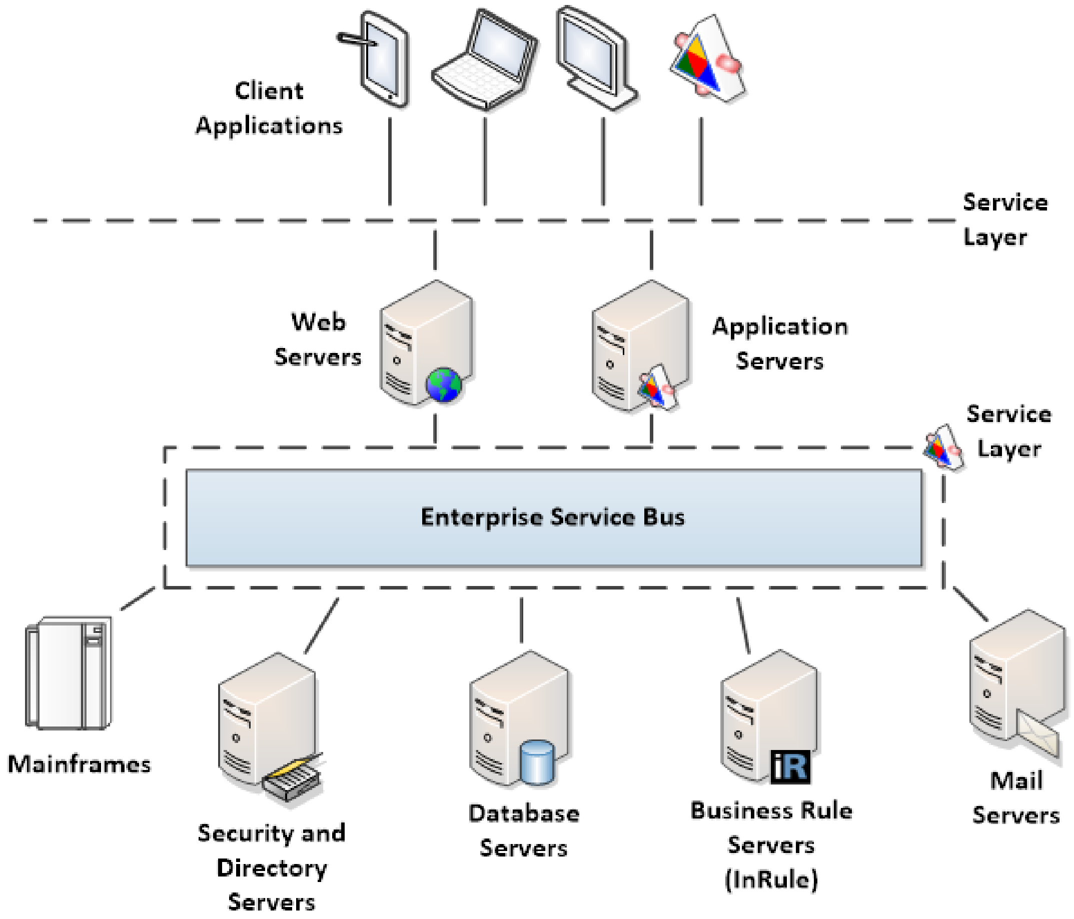


5.2.3 ESB and Messaging

Integrating Service-Oriented Business Rule Execution with an Enterprise Service Bus

Key Points

- Many ESB implementations require complex business rules to be integrated across several disparate systems
- Rule execution occurs only on the Business Rule Server(s)
- Rule execution can scale-out with demand by spreading load across a server farm
- A single, generic, rule-processing endpoint can be reused for many different types of messages for many different business problems
- XML or JSON state is passed in the request and the response usually contains modified state and/or related messaging
- In scale-out and high-availability scenarios, a load balancer can be used to spread message requests across a farm of servers
- InRule can be used to call back into other resources exposed by the service bus

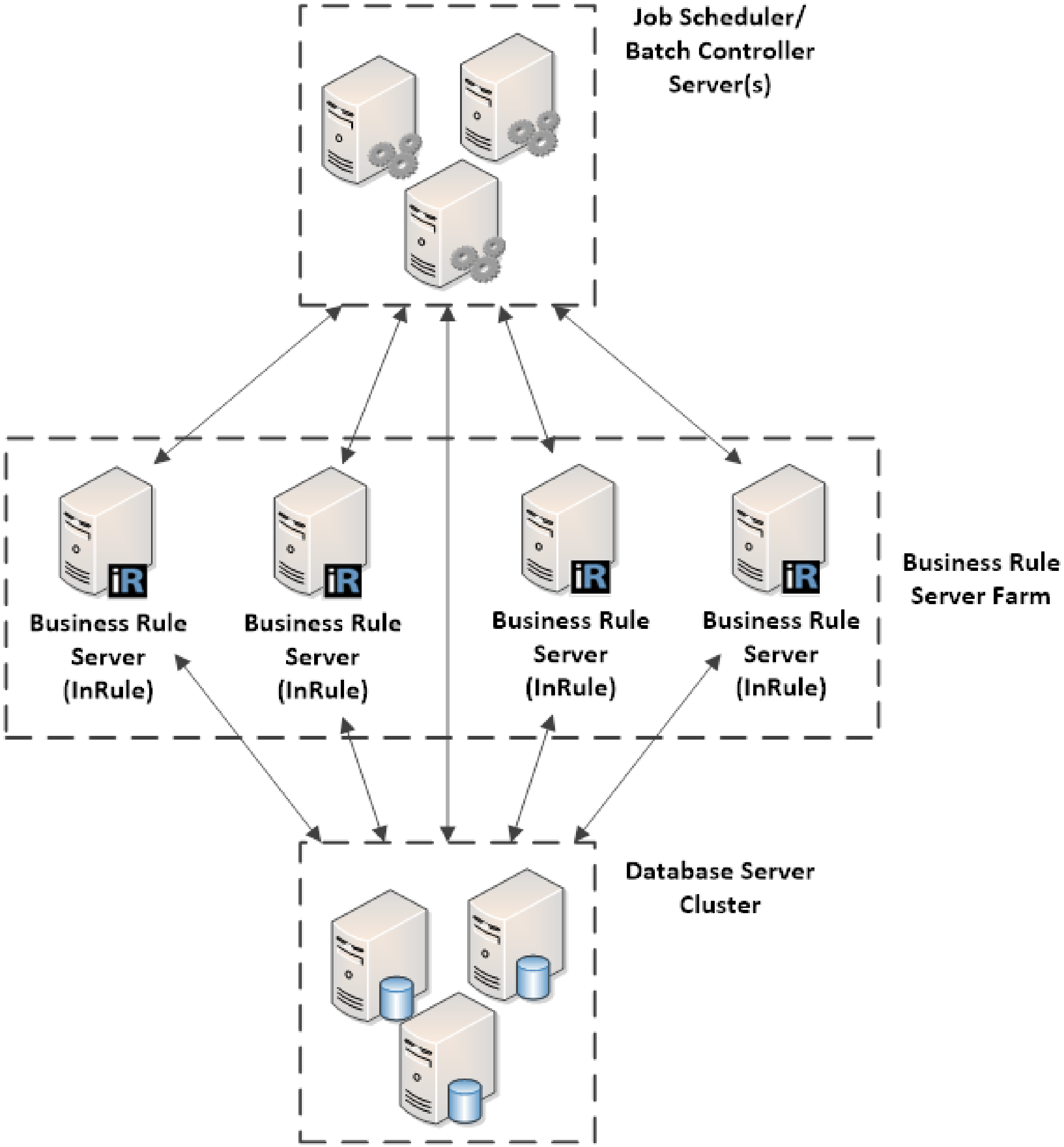


5.2.4 Job Scheduling and Batch Processing

Batch Processing with Rules

Key Points

- Job scheduler or batch processing application aggregates rule requests (initiated on a schedule or by a user)
- Requests are queued for rule processing
- Batch processing application spreads requests across a farm of business rule servers
- Stateless requests are multi-threaded across many servers and processors, lowering overall batch processing time
- Many solutions benefit from tools that help manage job scheduling, such as Quartz.NET

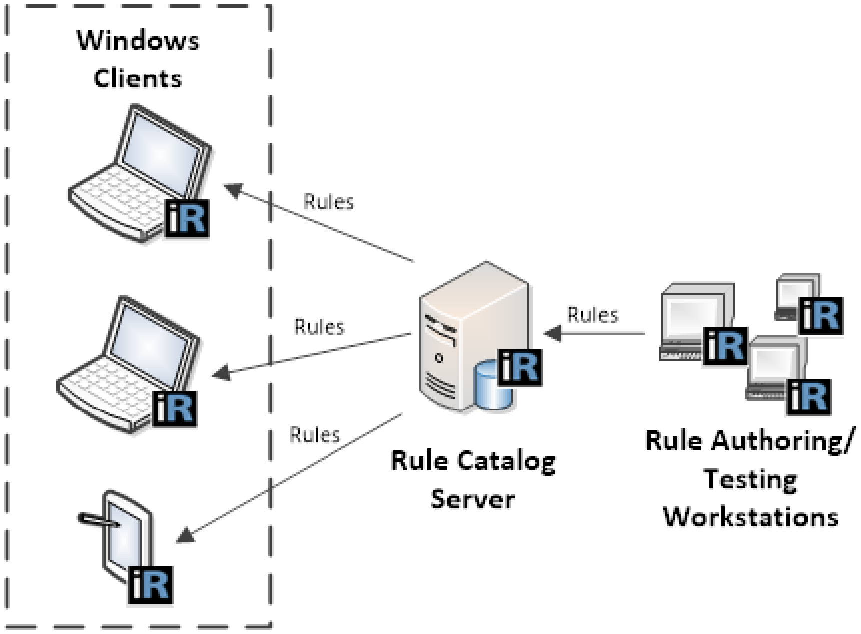

Job Scheduler/
Batch Controller
Server(s)
Business Rule Server (InRule)
Business Rule Server (InRule)
Business Rule Server (InRule)
Business Rule Server (InRule)
Business Rule
Server Farm
Database Server
Cluster

5.2.5 Desktop and Disconnected Applications

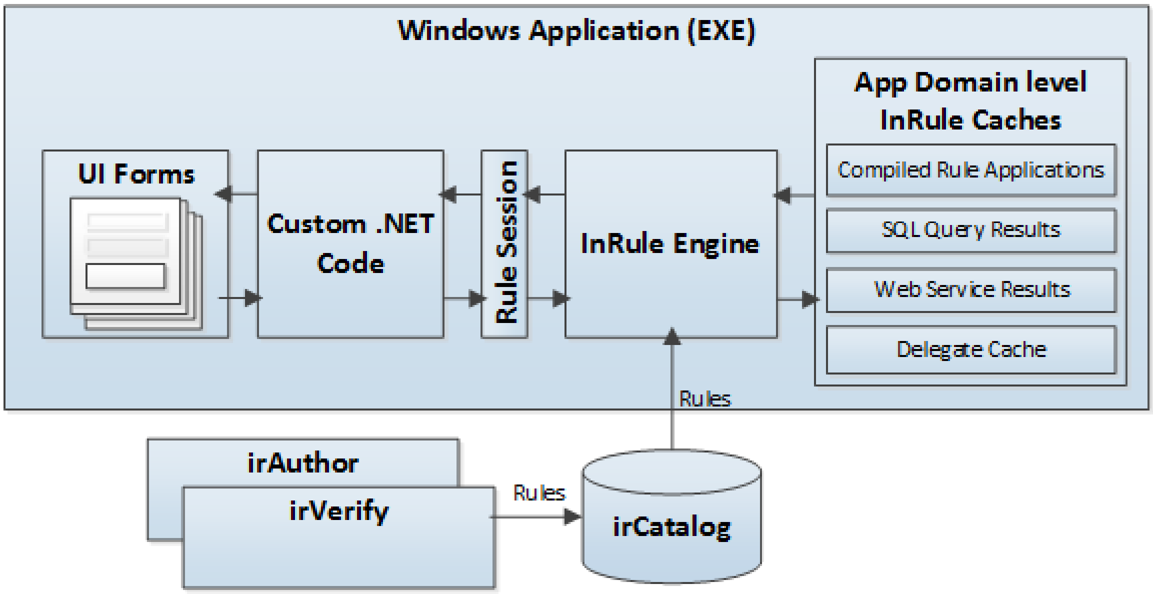
Running Rules locally on a desktop, laptop, or Windows device

Key Points

- Rule execution occurs on each client machine, with no requirement for a network connection
- The rules are stored on each machine as a small file
- The client application can use irSDK code to pull the latest copy of the rules from irCatalog when the machine is connected to the network.
- An instance of the rule engine is instantiated using irSDK code within the client application



A possible application architecture for a desktop Windows EXE that executes rules



5.3 Common Integration Scenarios

Overview

This section contains some sample approaches for implementing InRule on common technology stacks that are used with .NET applications. Please be aware that many other configurations are possible.

- Web Services (WCF)
- ASP.NET MVC

- BizTalk
- Microsoft® Dynamics® CRM
- SharePoint
- Entity Framework
- Enterprise Rule Services

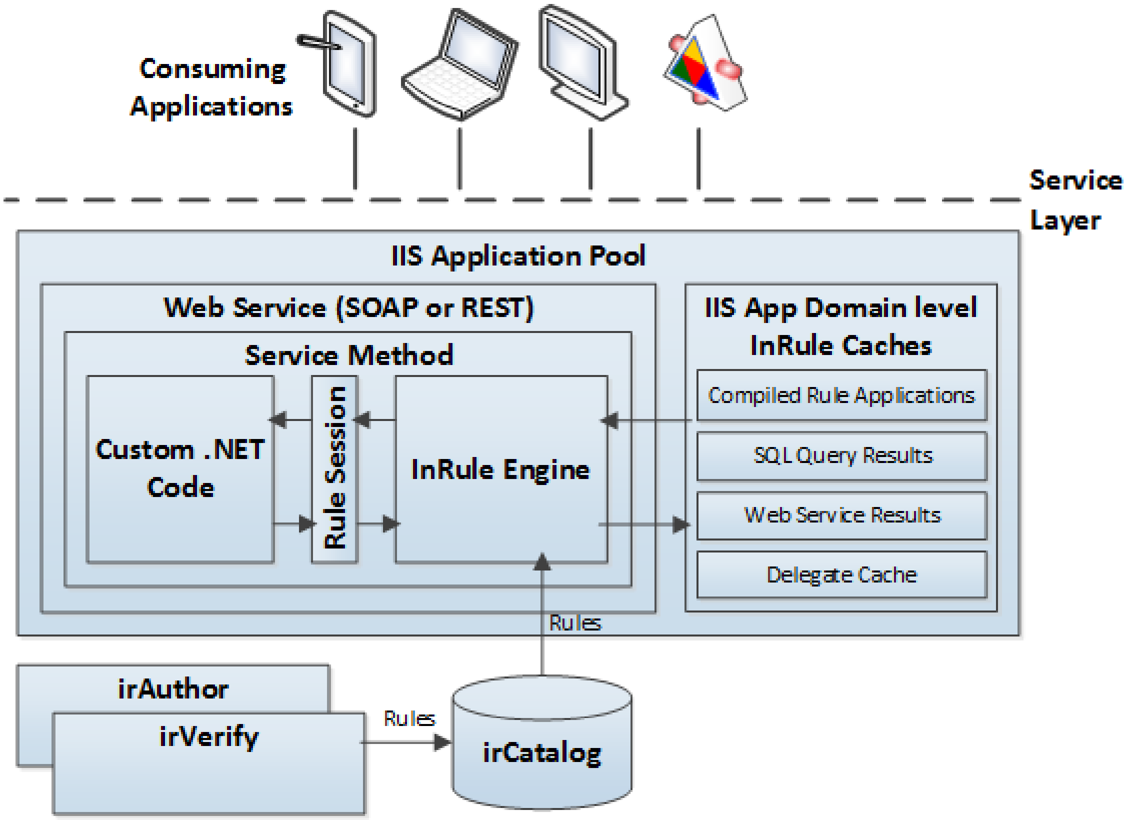
5.3.1 Web Services

Rule integration with Rules from a reusable service layer

Key Points

- WCF or Web API Service that hosts rule execution SDK code
- This is the most popular approach used to integrate InRule with customer applications
- A specialized interface can be provided for a specific business problem, or a generic interface can support many different sets of rules with many different data schemas
- Rule execution can scale-out with demand by spreading load across a server farm
- Many different consuming applications and platforms can reuse the same rule-processing web service interface
- IIS supports robust, configurable multi-threaded rule request processing
- InRule includes generic out-of-the-box SOAP and REST execution services (irServer® Rule Execution Service) with irServer that may be applicable for some solutions
- InRule provides a code sample for a generic REST execution service that can be customized to meet requirements for most solutions

A possible application architecture for a Web Service that executes rules



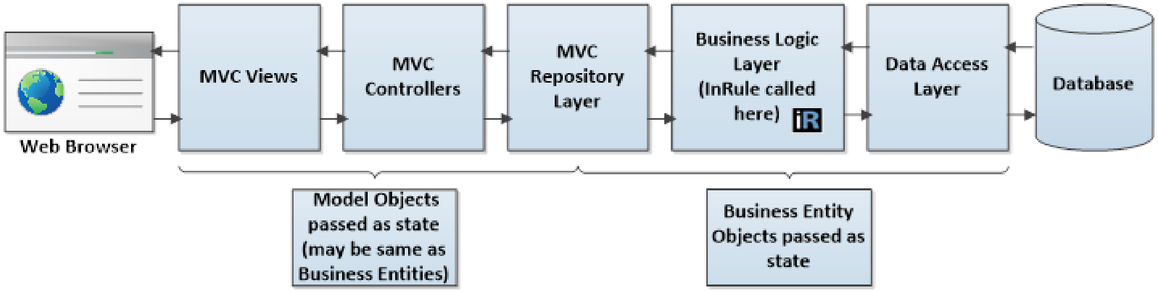
5.3.2 ASP.NET MVC

Integration with ASP.NET MVC web applications

Key Points

- InRule can run in-process on web servers
- In typical "layered" architectures, business logic is separated into its own component
- InRule can bind directly to .NET types, including types generated by the Entity Framework
- InRule can be used for data validation, and complex business logic operations

Using an ASP.NET MVC web application as an in-process host to execute rules


Web Browser
MVC Views
MVC Controllers
MVC Repository Layer
Business Logic Layer (InRule called here)
iR
Data Access Layer
Database
Model Objects passed as state (may be same as Business Entities)
Business Entity Objects passed as state

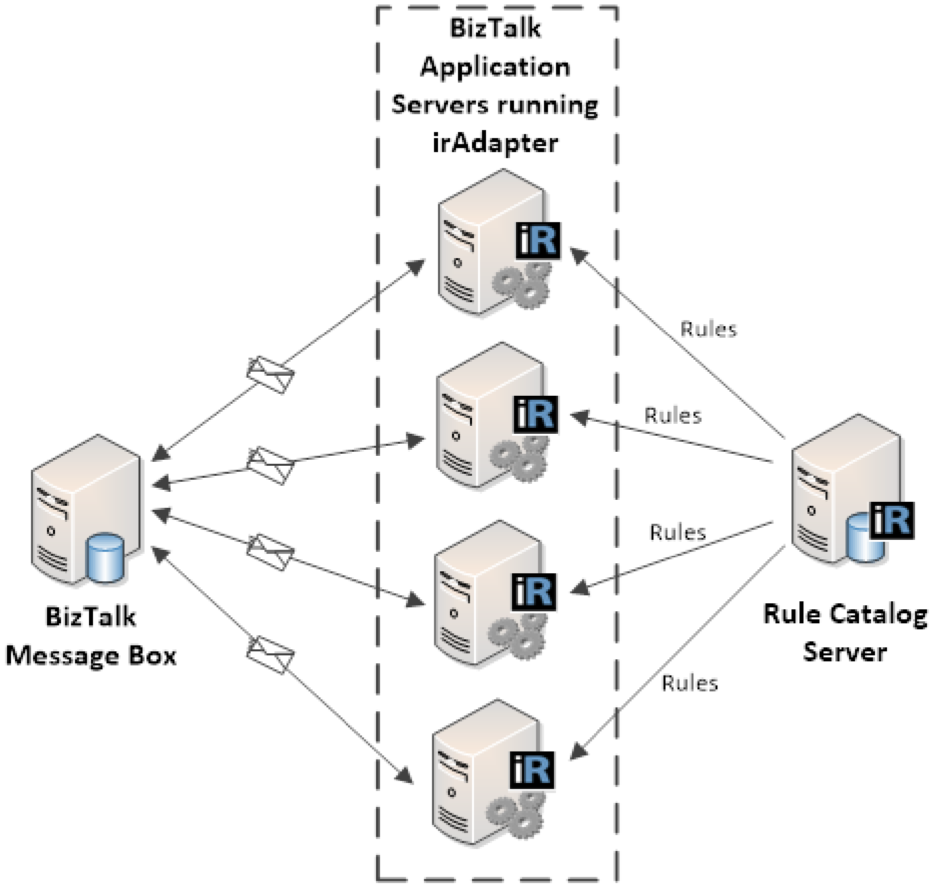
5.3.3 BizTalk

InRule with BizTalk

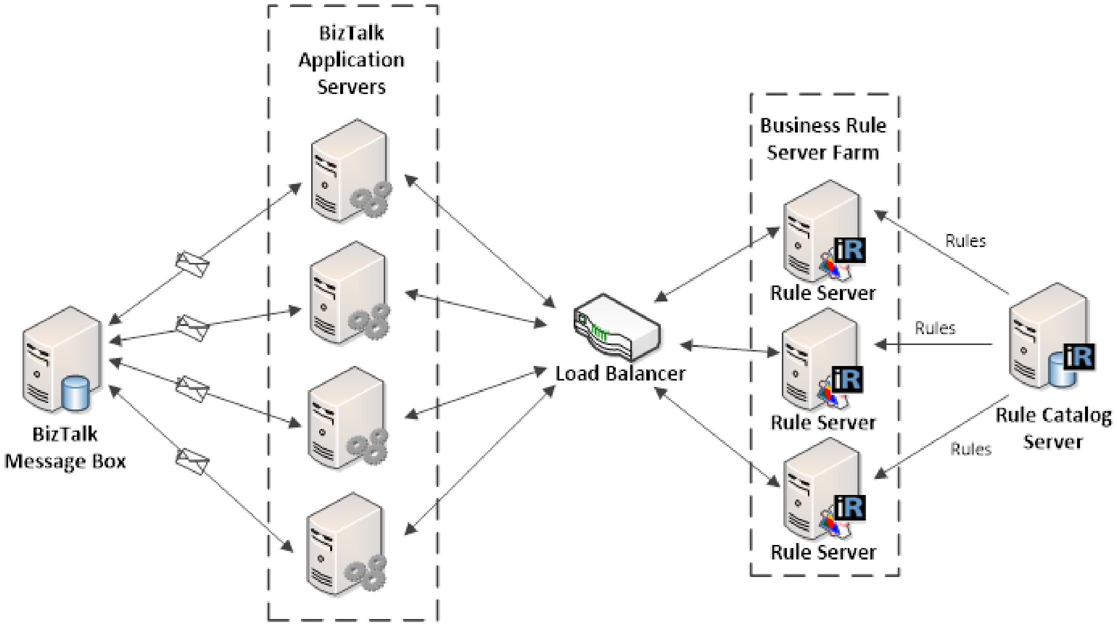
Key Points

- BizTalk provides a robust framework for batch processing and reliable messaging. However, many back-end processes require complex business logic for validation or transformation
- InRule ships with an adapter for BizTalk that can run within a BizTalk Host Instance
- InRule also ships with a DLL that can be called from a transform shape directly in an orchestration. Note that this approach is only suggested when rule request sizes are small (< 100K XML) and have short execution times (< 250 ms).
- For more advanced and scalable deployments, InRule can be hosted in a WCF service on a server farm, and process BizTalk messages sent using the web service adapters

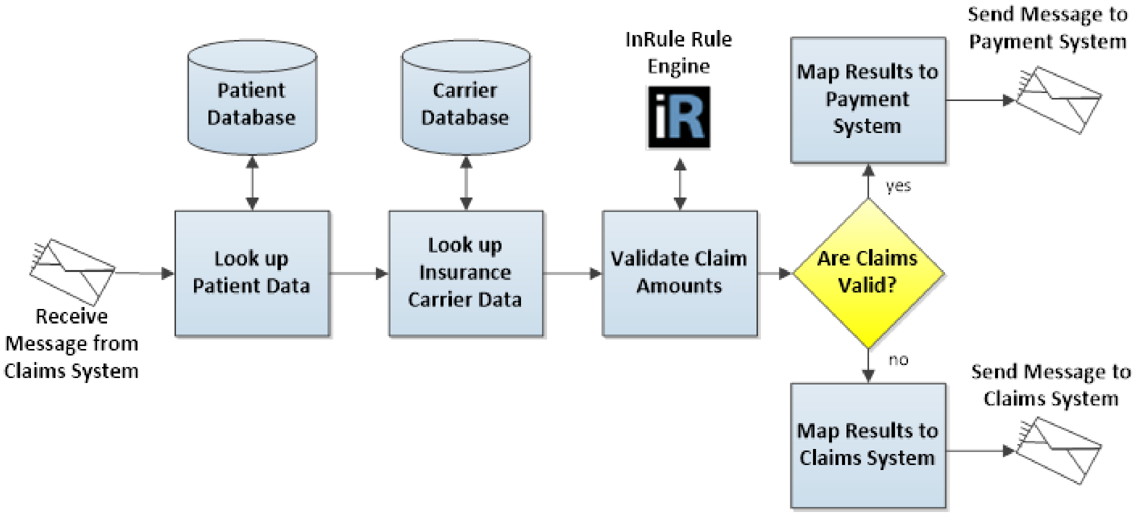
BizTalk message processing using the InRule adapter for BizTalk



BizTalk message processing using an external, scalable Business Rule Server farm



A sample BizTalk orchestration that processes health care claims. The rule engine is sent a message to perform complex business logic comparisons and calculations to ensure claims are valid.


Patient Database
Carrier Database
InRule Rule Engine
Map Results to Payment System
Send Message to Payment System
yes
Receive Message from Claims System
Look up Patient Data
Look up Insurance Carrier Data
Validate Claim Amounts
Are Claims Valid?
no
Map Results to Claims System
Send Message to Claims System

5.3.4 Dynamics CRM

Rule integration with Microsoft® Dynamics® CRM and rules from a reusable service layer

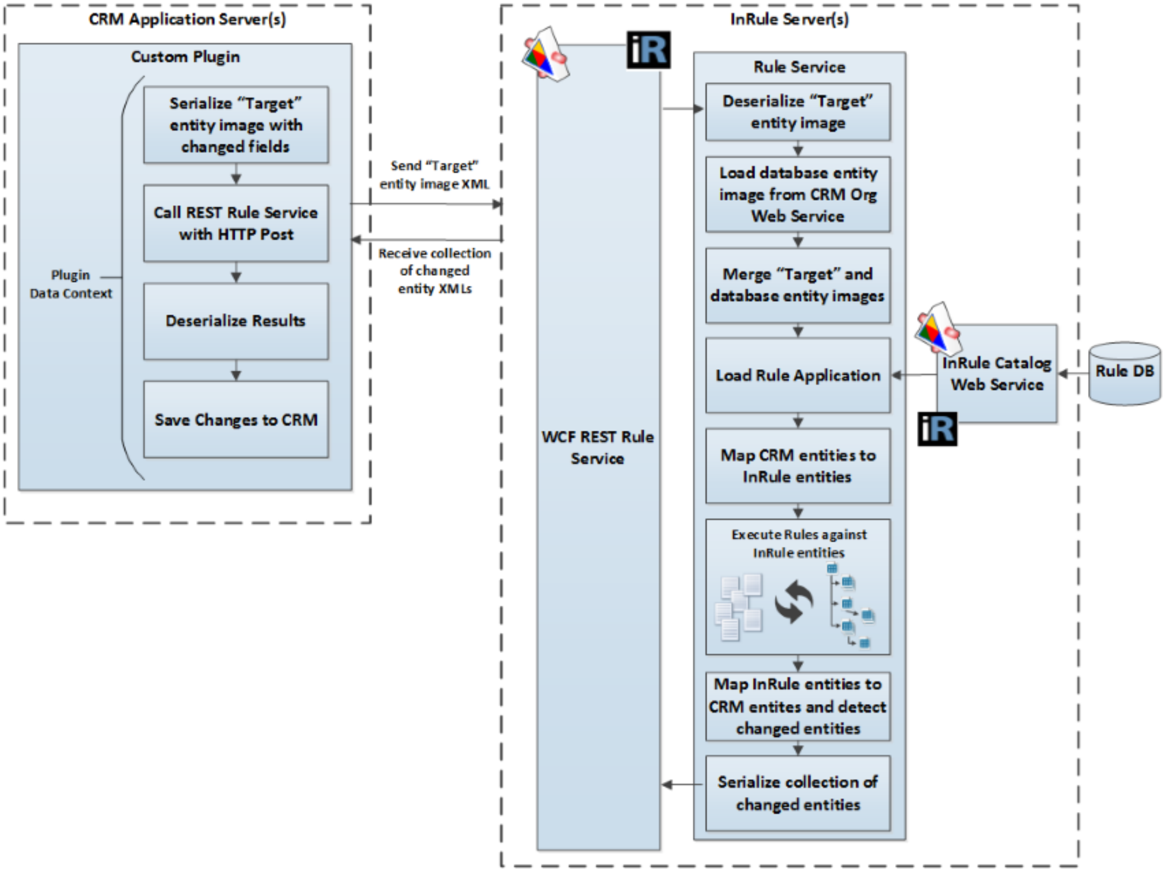
Key Points

- InRule has code samples available that demonstrate execution of rules against CRM entities
- For most on-premise scenarios, a light-weight plugin can run inside the Microsoft® Dynamics® 365 Sandbox and call a service that runs rules in a different process
- For on-line scenarios, a similar light-weight plugin approach can be applied when using CRM

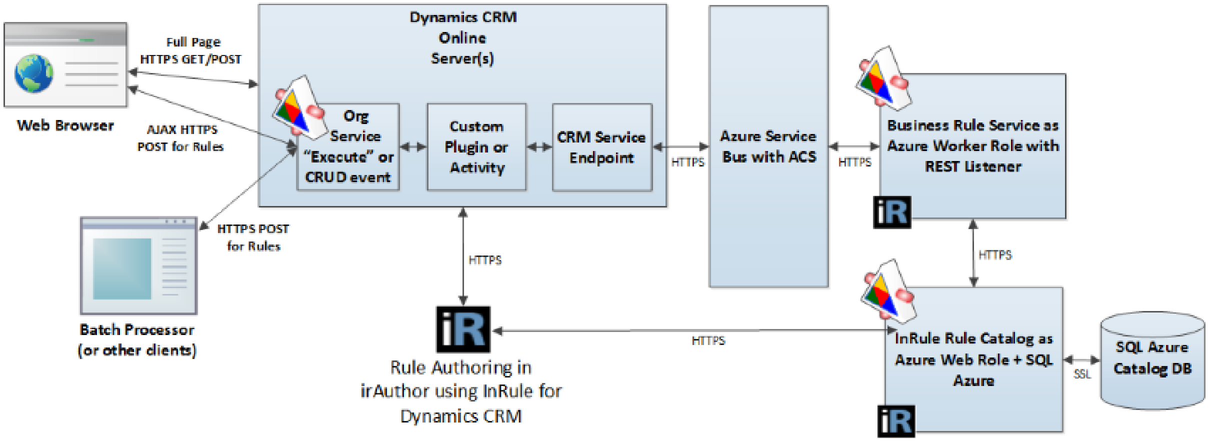
Service Endpoints and the Microsoft® Azure® Service Bus

- For out-of-process, cloud, and AJAX scenarios, a REST web service approach offers a simple, re-usable interface to process rules against CRM entities

Using the sample light-weight Sandbox Plugin code with REST service for InRule integration with Microsoft® Dyanmics® 365


CRM Application Server(s)
Custom Plugin
Serialize "Target" entity image with changed fields
Call REST Rule Service with HTTP Post
Deserialize Results
Save Changes to CRM
Plugin Data Context
Send "Target" entity image XML
Receive collection of changed entity XMLs
InRule Server(s)
WCF REST Rule Service
Rule Service
Deserialize "Target" entity image
Load database entity image from CRM Org Web Service
Merge "Target" and database entity images
Load Rule Application
Map CRM entities to InRule entities
Execute Rules against InRule entities
Map InRule entities to CRM entites and detect changed entities
Serialize collection of changed entities
InRule Catalog Web Service
Rule DB

For Microsoft® Dynamics® 365 Online, a sandbox plugin can use the Microsoft® Azure® Service Bus to execute rules using a Microsoft® Azure® worker role

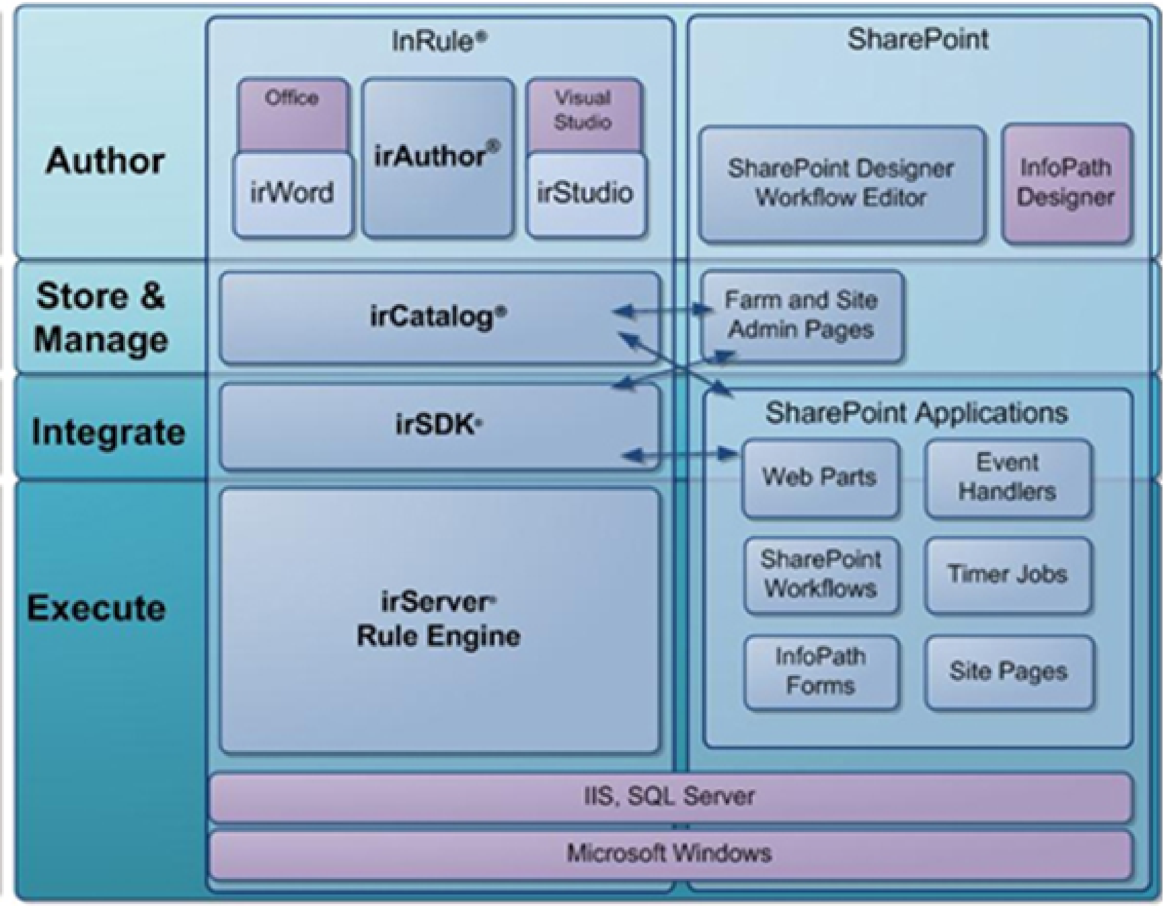


5.3.5 SharePoint

InRule for SharePoint

Key Points

- InRule can be called from event handlers and workflows
- Web components such as WebParts, and farm Pages can call InRule using code-behinds and AJAX
- Advanced implementations can include a reusable rule service that can be called the same way from many different SharePoint components
- InRule versions 4 and later requires the .NET 4 framework or later. Due to this limitation, InRule v4 cannot run in-process with SharePoint versions older than 2013. InRule suggests integrating with SharePoint using a set of out-of-process web services.



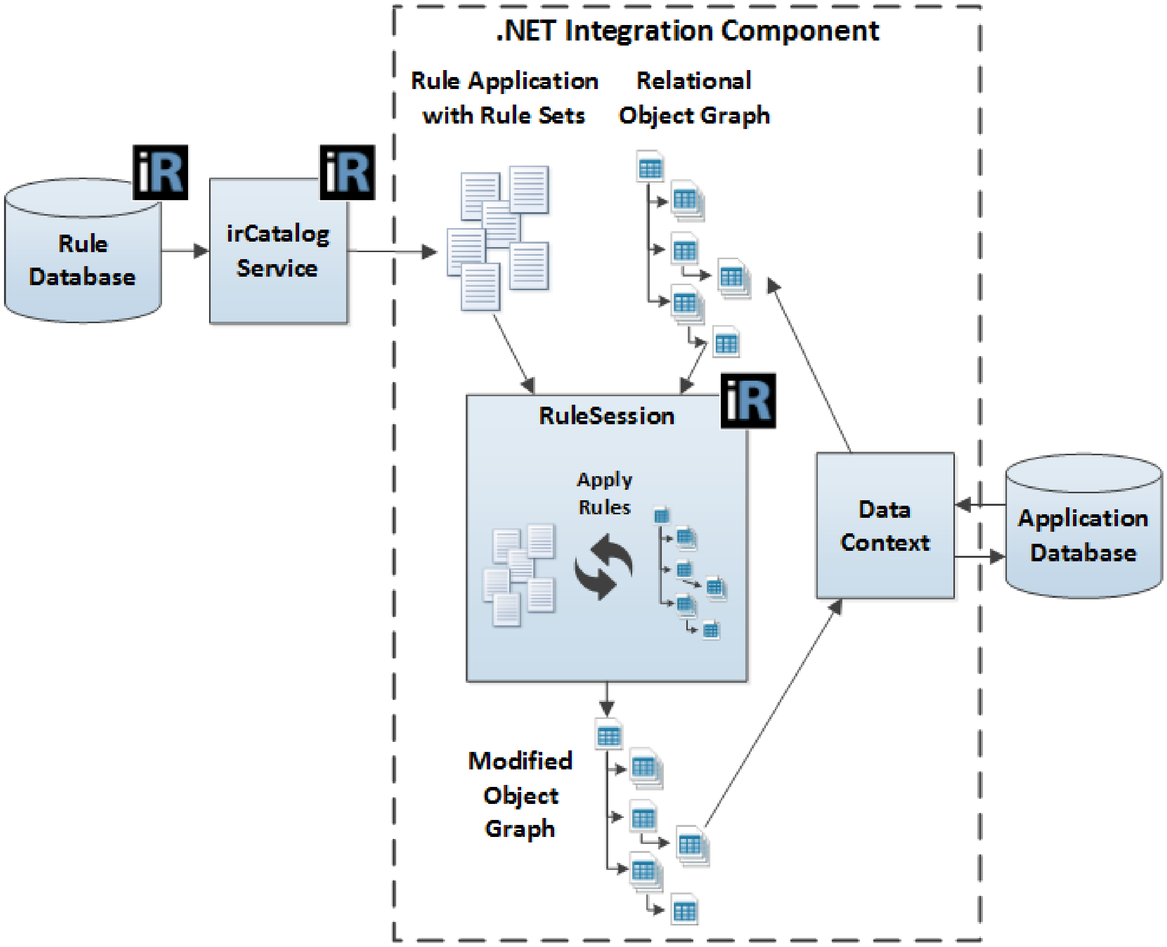
5.3.6 Entity Framework

Integration with Entity Framework and Object Relational Mappers (ORMs)

Key Points

- InRule entity schema is driven from .NET POCO objects written by hand or generated by coding tools
- Rules can operate directly against Entity Framework entity types that can be passed directly back to a data context after rule operations are completed
- Provides an efficient approach when rule application schemas must closely model database schemas- Entity Framework objects can be generated to match database table structures and then imported directly into rules
- Use the .NET Data Context for the most efficient approach for loading and saving entity data
- The Entity Framework default data operations can be overridden with stored procedures or custom SQL when the rule schema will not exactly match the data schema
- InRule Execute SQL Query actions can be used to query record sets and populate entity collections
- Any 1-to-Many relationships may be expressed using property types that implement IList, IList<T>, ICollection<T> or IEnumerable<T> interfaces

Using the Entity Framework to provide a schema and application data with rules that are stored in irCatalog

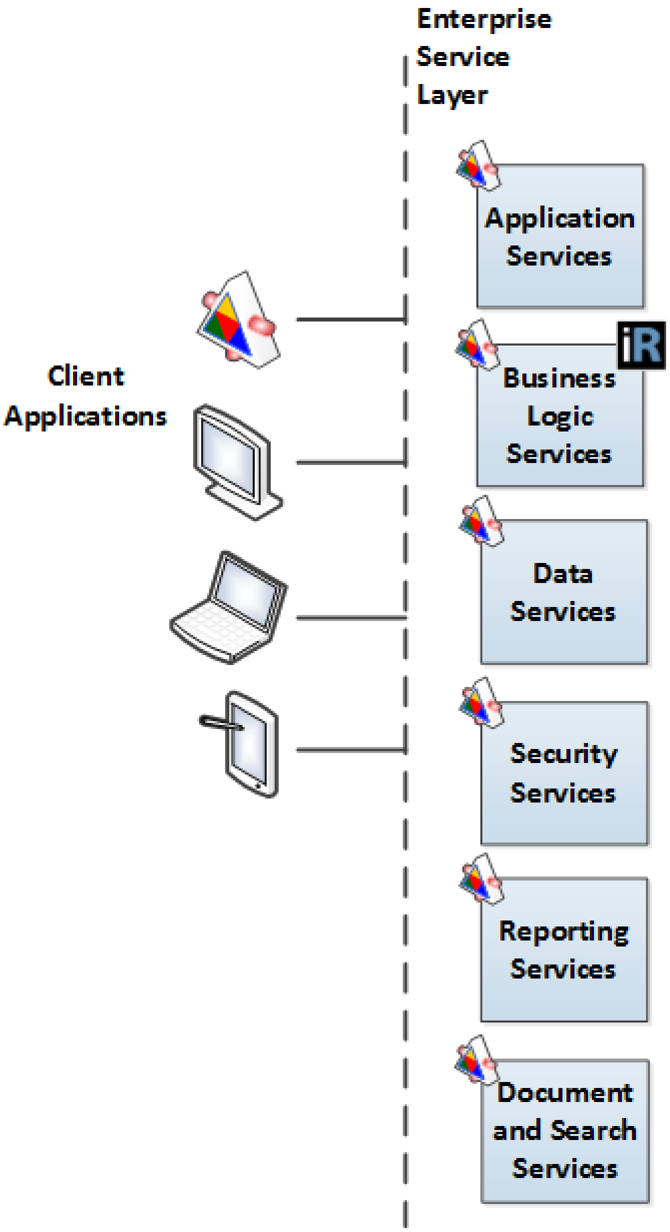


5.3.7 Enterprise Rule Services

Enterprise Rule Services

Key Points

- Many enterprises consider creating a set of atomic services that can be combined for various permutations of business processes and consuming application clients.
- Create business logic services that run InRule in-process using REST or SOAP interfaces. These custom rule execution services can conform to enterprise standards for interface, security, logging, and wire protocols.
- A specialized interface can be provided for a specific business problem, or a generic interface can support many different sets of rules with many different data schemas
- Rule execution can scale-out with demand by spreading load across a server farm, and can be hosted on premise or in the cloud
- Using REST or simple SOAP interfaces, other enterprise services can call into business logic services, or business logic services can consume other services during rule execution



5.4 Rule Deployment

As changes are made to rule applications, the changes need to be managed across development, testing, and production environments. Each InRule implementation should include a well-thought strategy for propagating rule changes with predictable processes. Although the flexibility of the InRule SDK and rule application file formats allows for many possible strategies, InRule suggests the following standard approaches for rule life-cycle management.

- Rule Management with Multiple Catalogs
- Rule Management with Labels
- Rule Management Hybrid

- Rule Management with Files

The physical design of the irCatalog deployment should be considered along with the logical design for maintaining rules. Two possible approaches are described in the pages below.

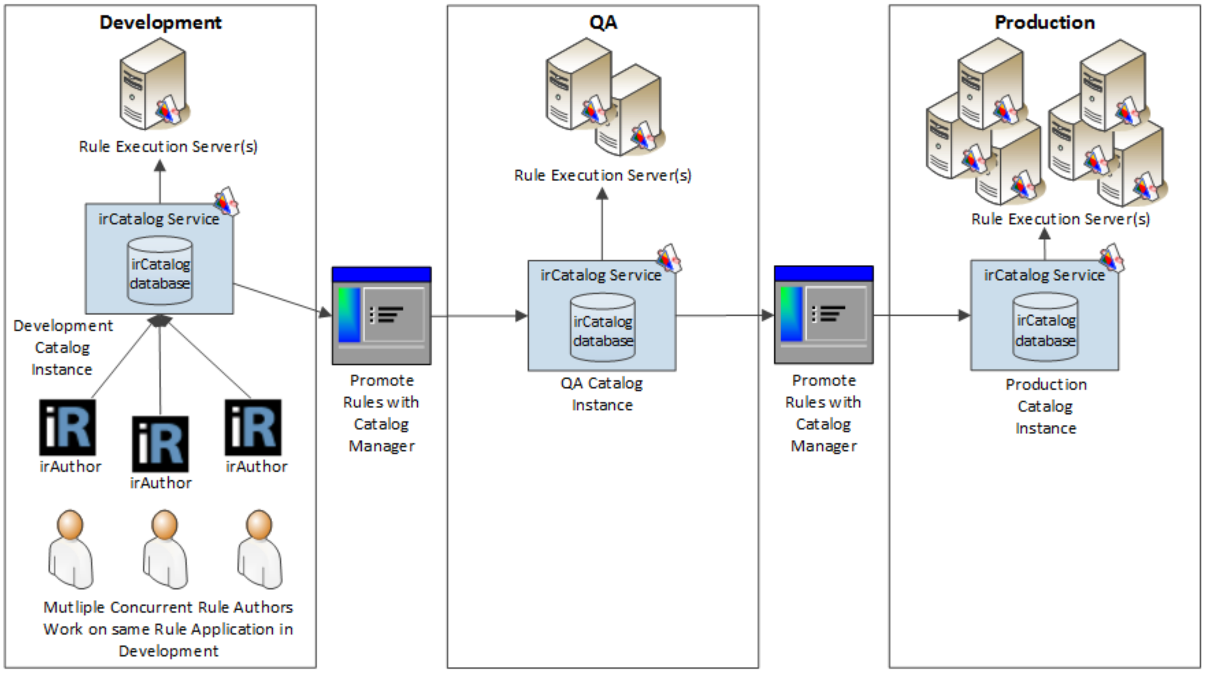
- Catalog Deployment with Co-located Services
- Catalog Deployment with Separate Servers

.

5.4.1 Rule Management with Multiple Catalogs

Multiple Catalogs for the Rule Development Life Cycle

- Each environment requires an independent copy of the Catalog web service and database
- Rule changes can be propagated using the promotion feature in the Catalog Manager
- Rule changes can also be propagated using the promotion method in the InRule SDK
- If elements are shared in development, the sharing links are not maintained across catalog instances
- Multiple rule authors can edit the same rule application concurrently
- Rule changes can be hot-deployed from the Catalog web service without restarting production application
- Production application is dependent on stability of production Catalog web service and database
- Full rule application revision audit trail is available in each environment

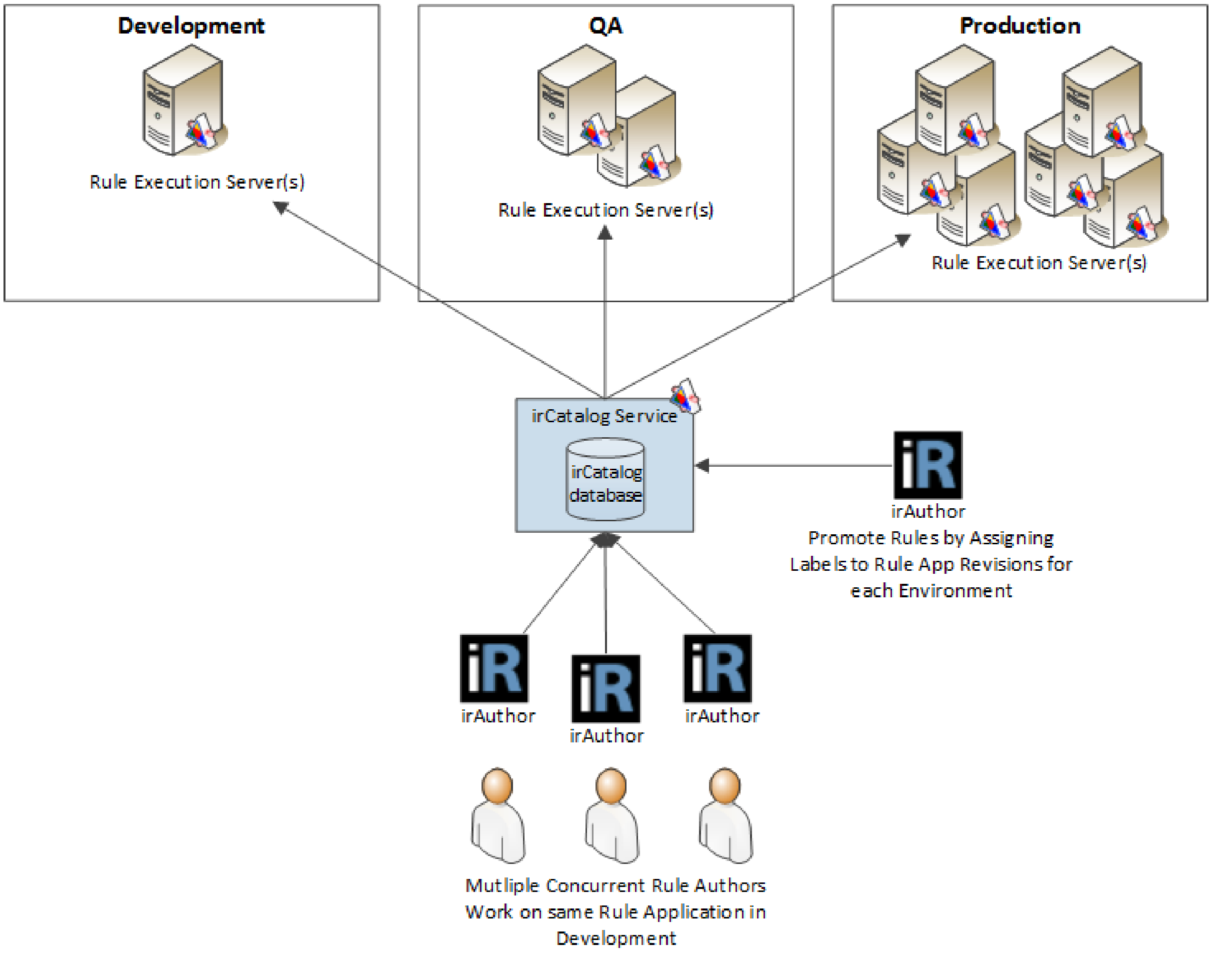


Promotion is intended to run in a linear path, such as Development to QA to Production, as in this example. Backward or circular promotion paths are not supported.

5.4.2 Rule Management with Labels

Using Revision Labels to Manage the Rule Development Life Cycle

- All environments share a single copy of the Catalog web service and database
- Rule changes can be propagated by assigning labels to rule application revisions in irAuthor or with irSDK
- If elements are shared in development, they are also shared in QA and Production
- Multiple rule authors can edit the same rule application concurrently
- Rule changes can be hot-deployed to the Catalog web service without restarting production application
- Production application is dependent on stability of the shared Catalog web service and database
- Full rule application revision audit trail is available in each environment



.

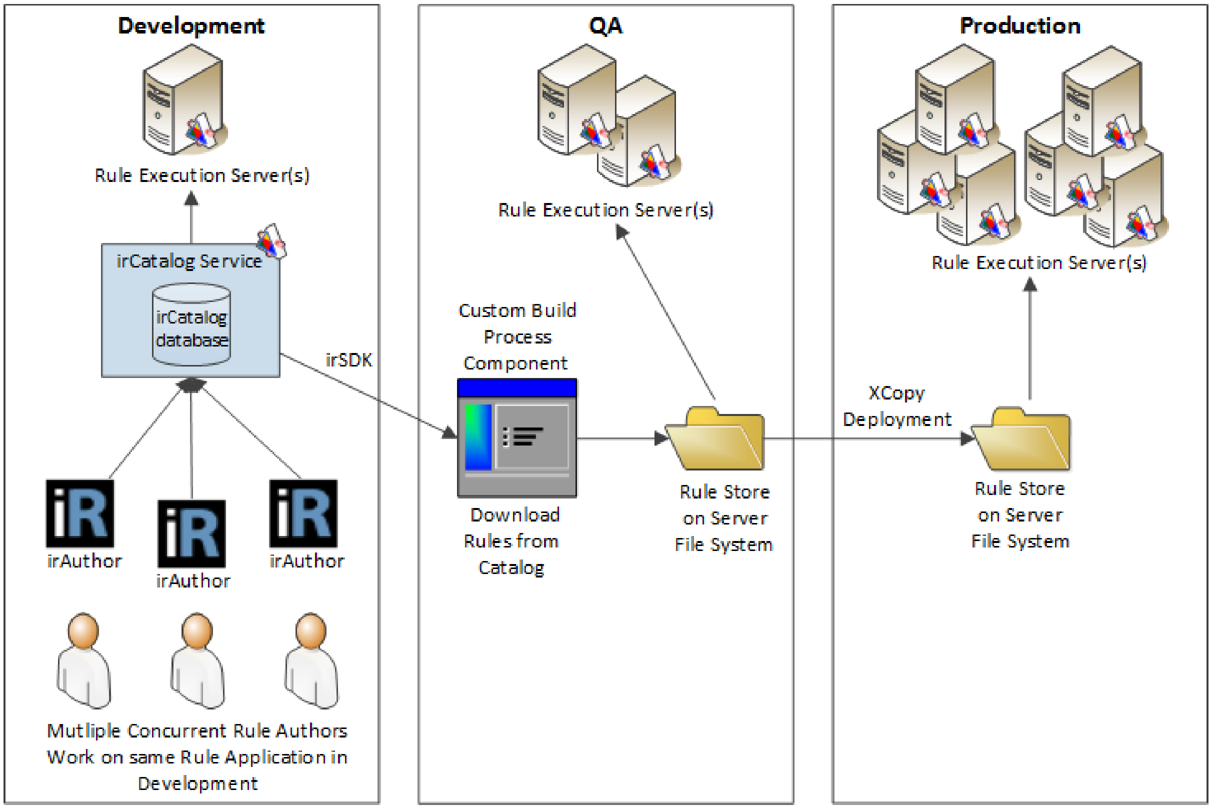
5.4.3 Rule Management Hybrid

Using the Catalog only in Development for the Rule Development Life Cycle

- The Catalog is used in development and the file system is used for rules in other environments
- Only one Catalog web service and database instance is required, for use by the development environment
- Rule changes can be propagated by downloading rules from the catalog web service and then

saving to the file system in other environments

- Multiple rule authors can edit the same rule application concurrently
- Rule changes can be hot-deployed from the production file system without restarting production application
- Production application is not dependent on stability of the Catalog web service and database
- Audit trail for rule changes is only available in development
- A simple, custom-coded component may be required for integration with automated build processes or continuous integration

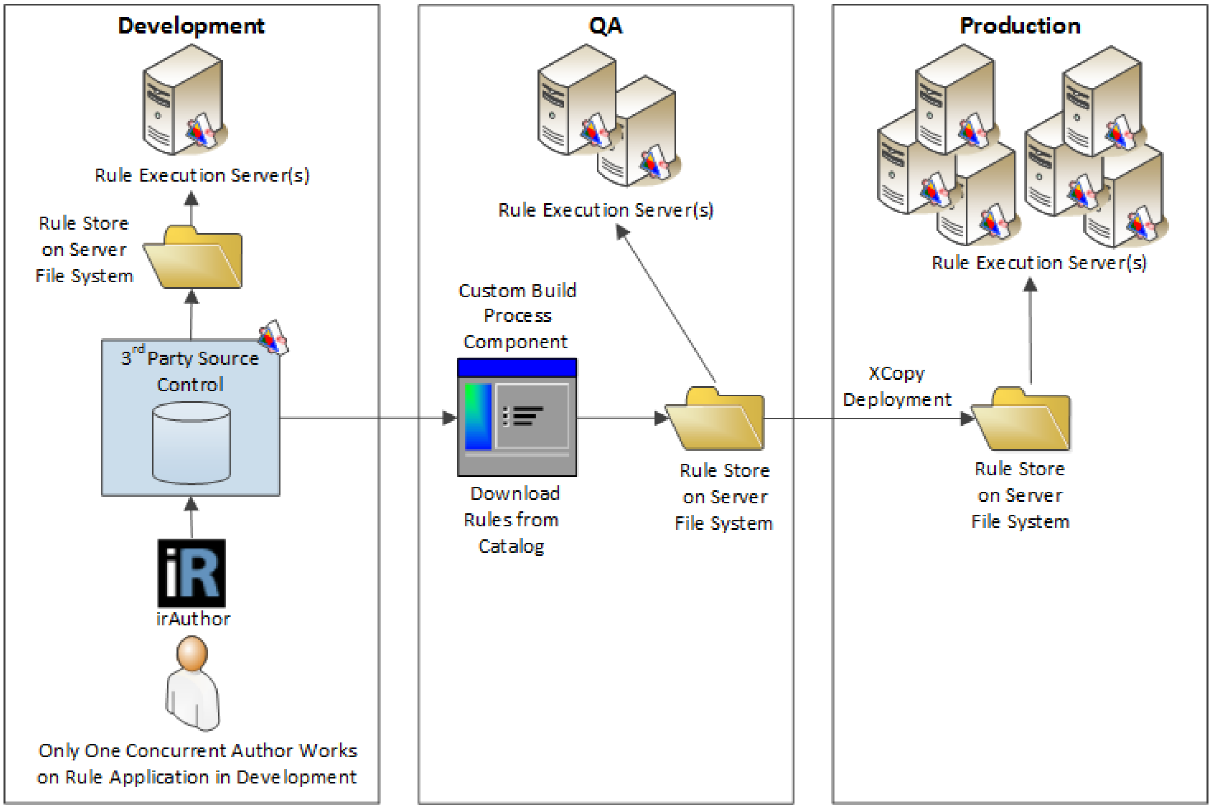

Development
Rule Execution Server(s)
irCatalog Service
irCatalog database
irSDK
irAuthor
irAuthor
irAuthor
Mutliple Concurrent Rule Authors Work on same Rule Application in Development
QA
Rule Execution Server(s)
Custom Build Process Component
Download Rules from Catalog
Rule Store on Server File System
Production
Rule Execution Server(s)
XCopy Deployment
Rule Store on Server File System

.

5.4.4 Rule Management with Files

Using the File System for the Rule Development Life Cycle

- No Catalog components are required (using a source control system is highly recommended)
- Rules are managed the same way as any other application source file
- Rule changes are propagated by copying rule application files across environments
- Only one rule author can edit the same rule application concurrently
- Rule changes can be hot-deployed from the production file system without restarting production application
- Production application is not dependent on stability of the Catalog web service and database
- Audit trail for rule changes is limited to whatever 3rd party source control system provides and is only available in development
- A simple, custom-coded component may be required for integration with automated build processes or continuous integration


Development
Rule Execution Server(s)
Rule Store on Server File System
3rd Party Source Control
irAuthor
Only One Concurrent Author Works on Rule Application in Development
QA
Rule Execution Server(s)
Custom Build Process Component
Download Rules from Catalog
Rule Store on Server File System
XCopy Deployment
Production
Rule Execution Server(s)
Rule Store on Server File System

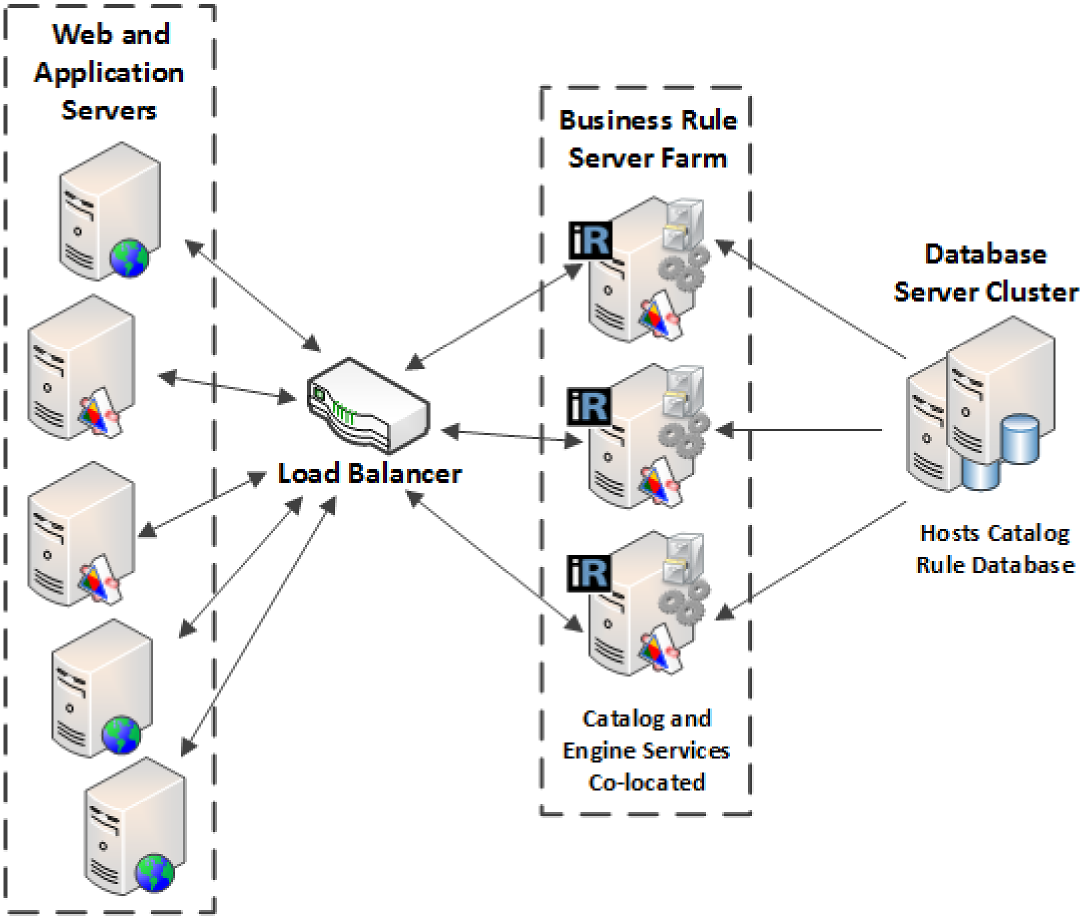
.

5.4.5 Catalog Deployment with Co-located Services

Deploying the Catalog using the same servers as Rule Execution

- Catalog Services can reuse hardware and VMs that have been deployed for Rule Execution
- All Catalog Services are configured to point to the same rule catalog database using configuration files
- Each instance of the Rule Execution service is configured to point to the Catalog Service instance that is co-located on the same machine

- For IIS deployments, InRule suggests hosting the co-located Rule Execution and Catalog Services in separate Application Pools
- Load balancer passes requests across HTTP rule execution services

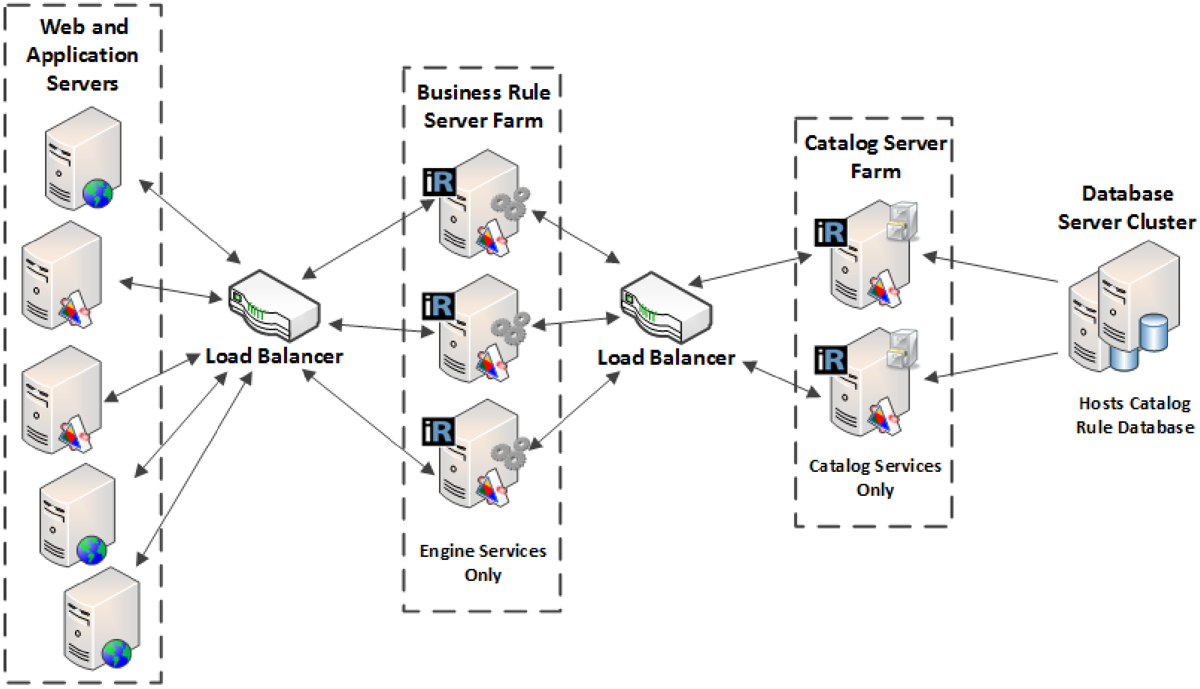

Web and Application Servers
Business Rule Server Farm
Database Server Cluster
Load Balancer
Hosts Catalog Rule Database
Catalog and Engine Services Co-located

.

5.4.6 Catalog Deployment with Separate Servers

Deploying the Catalog using Separate Servers for irCatalog

- Catalog services are scaled and maintained on a separate set of servers
- The Catalog usually does not need to scale out due to performance concerns, but instead to provide high availability
- The Rule Execution load balancer passes requests across HTTP rule execution services
- All Rule Execution services are configured to point to the same Catalog load balancer URL
- All Catalog servers are configured to point to the same rule database using configuration files
- Separate Catalog servers may require separate license activations



.

5.5 Performance Tuning and Best Practices

Overview

This section contains additional miscellaneous information about product configuration, performance tuning, and best practices.

- Multi-threaded Rule Execution
- Cold Start Mitigation
- Managing the RuleApp Cache
- Event Log
- Rule Engine Execution Log
- Hosting the Catalog on IIS vs Windows Services

5.5.1 Multi-threaded Rule Execution

In high-volume applications that make numerous calls to the rule engine, using a multi-threaded approach can help achieve better overall performance for batch processing. Take the following scenario as an example:

- A data transformation application processes approximately 100,000 records with InRule every night as a batch

- The application will run on a machine that has more than two processing cores

The InRule rule engine and catalog are designed for free-threaded execution. In the free-threaded model, memory caches are reused between rule engine instances running in different threads, which can lower the overall memory footprint and execution times for an application.

When InRule is run on multiple threads, the owner process can execute rule engine requests concurrently. Each processor core shares the load in processing rules, which can produce an overall execution time that is significantly lower than if the entire batch was executed serially in a single thread on a single core.

With any multi-threaded application, there is an inherent performance overhead incurred for each new thread that is introduced into the process. With a .NET application, this overhead can be exacerbated by the need to periodically run garbage collection across a heap of memory that is shared between threads. During garbage collection, the .NET runtime blocks executing threads so that the memory heap can be examined and cleaned in a thread-safe manner.

Since InRule is built from the ground-up in .NET, it experiences the same scalability concerns inherent with any .NET application that must work with a large number of .NET objects. In many cases, adding more concurrent execution threads beyond one thread per core to these types of applications will not improve overall performance.

Threading Guidelines:

Given that each rule application is unique with respect to both the complexity and depth of its schemas and rulesets, it is difficult to make a blanket statement about the best threading model to use for all applications that are running the rule engine. Applications with a large number of rules may perform better when more threads are added, while large state models may quickly reach a point of diminishing returns as new threads are added.

Although optimal threading models vary greatly between applications, the following guidelines are presented as general best-practices:

- **Use a maximum of one rule execution thread per processor core --** InRule is processor intensive, so adding more threads beyond one per core generally does not help performance. In addition, a large number of threads may increase contention for memory caches and garbage collection thereby degrading performance.
- **Run services under IIS instead of a Windows Service --** IIS has optimized request handling for large numbers of requests across threads.
- **Set the Garbage Collection mode for the .NET runtime to "gcServer" --** By default the .NET runtime will use "gcWorkstation", which optimizes garbage collection for UI applications and UI background threading. The "gcServer" is recommended for machines that have more than two logical processors and will be running free-threaded applications. By default an IIS install should already have processes set to use gcServer, but this setting may need to be manually adjusted for other .NET applications. For more information about adjusting this setting in a .NET configuration file, see .NET Framework Runtime Config Settings With InRule. More information about garbage collection modes is available from Microsoft on-line at https://msdn.microsoft.com/en-us/library/ms229357.aspx.
- **Run performance load tests before deploying to production --** To ensure that production load requirements are met, run soak tests on similar hardware to better anticipate real production throughput. Adjust threading models to optimize for the highest throughput.
- **Scaling across cores does not correlate to a linear improvement in performance --** The application overhead of running on more than one processor results in some loss of throughput as each new thread is included. For example, running an application on two threads may result in an overall improvement of 1.9 times instead of a linear factor of 2.0. This overhead increases as more cores are included, so that an application running on four cores may only show a factor of 3.5 times improvement over one thread on one core. Since each rule application can include a vastly different combination of rules an schema objects, these scalability factors can vary between implementations.
- **Scale "out" is preferred over scale "up"** -- To make the most efficient use of hardware, InRule suggests no more than eight physical cores per server. If rule processing demand exceeds the capacity of four to eight cores, then the option of configuring multiple four or

eight core servers in a farm should be explored before using a server with more than eight cores. A four-core server is considered the ideal sizing for most applications.

Configuration of IIS for Multi-Threading:

Given the popularity and robust nature of IIS on Windows Servers, many InRule implementations are hosted as services or applications running in IIS Application Pools. Each IIS AppPool contains AppDomains with .NET memory heaps. If too many threads are running against the same shared memory heap, then the application may experience a drop in throughput due to contentions. When running under IIS, the following guidelines may help optimize throughput:

- **Scale out physical servers over eight cores** -- If not using virtual machines, then load should be spread across physical servers that have a maximum of eight cores.
- **Use multiple VMs with a smaller number of cores assigned to each VM** -- If a hypervisor is available, run multiple VMs with two, four, or eight cores assigned to the VM. Spreading the load across multiple VMs generally results in less performance loss due to thread contention.
- **If the number of servers is limited, enable "web gardens" in IIS** -- Each Application Pool in IIS can be configured to run as one or more processes (the default is one). By setting the Maximum Worker Processes to greater than one, IIS with spread the request load across more than one process, which in turn spreads the load over more than one shared memory heap. This may help reduce threading contentions and improve throughput. Note that when more than one worker process is used, there is a significant increase in fixed memory cost, since each process must maintain a separate set of shared InRule caches.

5.5.2 Cold Start Mitigation

Rule Engine Cold Start

The first time a rule application is retrieved and executed, it must be compiled and loaded into memory. The time it takes for this process to complete is commonly referred to as the cold start. The cold start time will vary depending primarily on the size of the rule application. After the initial cold start, performance will more adequately represent normal rule engine execution performance.

Mitigating the cold start

A common practice for mitigating the cold start is to force the rules to compile before an end user makes a request for the rule application. This can be achieved when the application or service is loading.

The following SDK code options will force the rule application to begin compile:

1. The first creation of a InRule.Runtime.Entity

```
// Create rule application reference and a dummy Entity -- this performs
a Metadata compile, but not a Function compile
var ruleApp = new FileSystemRuleApplicationReference(@"C:\RuleApps
\MyRuleApplication.ruleappp");

using (var session = new RuleSession(ruleApp))
{
    session.CreateEntity("MyEntity");
}
```

2. Proactively call the Compile method on the RuleApplicationReference

```
// Create rule application reference
var ruleApp = new FileSystemRuleApplicationReference(@"C:\RuleApps
\MyRuleApplication.ruleappp");

// Force a Metadata compile - Function compile will still occur
incrementally and on-demand
ruleAppRef.Compile(CompileSettings.Create(EngineLogOptions.Execution));
```

Note: When performing a pre-compile as demonstrated above, it is important to set the LogOptions in the CompileSettings to the same value that will be used for execution. If the settings do not match, the first execution will include additional compile time.

There are two main steps to complete a full compilation of a rule application in InRule:

- **Metadata Compile** - Includes compile of the schema, endpoints, and lists along with rule validation
- **Function Compile** - Includes compile of rules and state access delegates

By default, the Metadata Compile is always completed the first time a rule application is used by a RuleSession or an explicit call is made to the RuleApplicationReference.Compile method. The Function Compile occurs incrementally and on-demand the first time a particular rule or state accessor is required by a particular rule execution request. In some scenarios, it may be desirable to force the entire Function Compile to occur upfront. To enforce a full upfront compile of a rule application, explicitly pass a CompileSettings instance to the call to RuleApplicationReference.Compile. See the code sample below:

```
          // Create rule application reference
          var ruleApp = new FileSystemRuleApplicationReference(@"C:\RuleApps
\MyRuleApplication.ruleapp");

          // Force a full Metadata and Function compile
          ruleApp.Compile(CacheRetention.Default,
CompileSettings.Create(EngineLogOptions.None));
```

There are various techniques for masking cold start mitigation in your rules-enabled application. Following are some of the more commonly used techniques when adding the cold start mitigation code to the 'on load' or 'on start' events of your application:

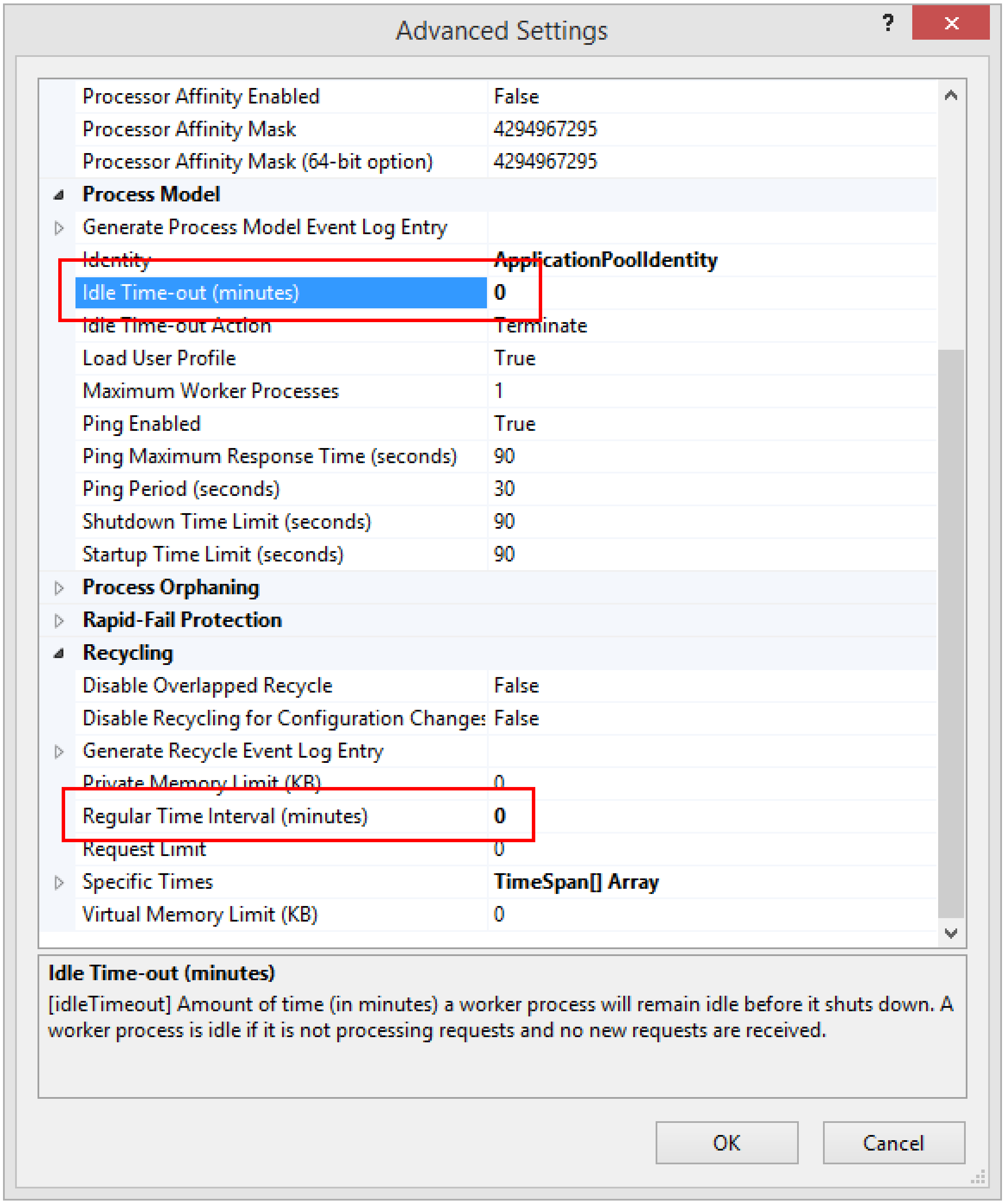
- Handle the cold start with an asynchronous worker.
- Display a splash screen if dealing with an end-user application.
- Consider disabling buttons or links that engage rule execution until the cold start mitigation is completed.
- For services, consider adding the above mentioned code to an event when the service is started (e.g. APPLICATION_ONSTART, IProcessHostPreloadClient).
- For ASP.NET or WCF applications, implement a startup routine in the Global.asax file.
- Use IIS or Windows Server AppFabric to auto-start services after an Application Pool is recycled

Helping minimize cold start with XmlSerializers

The first time rules are applied in a given AppDomain, the .NET XmlSerializer will attempt to create a temporary XML serializer assembly to perform an XML deserialization of a rule application. Once the temporary serializer assembly is generated and compiled, it is reused for the life of the AppDomain. The time to generate the serializer assembly can be significant, and can avoided by deploying a pre-built assembly that InRule provides as part of irSDK. The file named InRule.Repository.XmlSerializers.dll should be copied into the working directory of an application that is consuming InRule to avoid playing the cold start times for regenerating this XML serializer assembly for each new AppDomain.

Configuring IIS to minimize unanticipated cold starts

When InRule is hosted inside of an IIS worker process, it is important to note that IIS has built-in features that will occasionally recycle the worker process. Each time the process is recycled, all of the InRule AppDomain caches are also recycled, and cold start costs due to loading assemblies and compiling rule applications must be repaid. Two settings that affect automatic recycling of IIS application pools are the "Idle Time-out" and "Regular Time Interval". Both of these settings should be set to "zero" to avoid regular recycling of the process that is hosting both the InRule rule engine and the irCatalog. The screen shot below shows an example of configuring these settings using the IIS application pool Advanced Settings screen.



Processor Affinity Enabled	False
Processor Affinity Mask	4294967295
Processor Affinity Mask (64-bit option)	4294967295
◢ **Process Model**	
▷ Generate Process Model Event Log Entry	
Identity	**ApplicationPoolIdentity**
Idle Time-out (minutes)	**0**
Idle Time-out Action	Terminate
Load User Profile	True
Maximum Worker Processes	1
Ping Enabled	True
Ping Maximum Response Time (seconds)	90
Ping Period (seconds)	30
Shutdown Time Limit (seconds)	90
Startup Time Limit (seconds)	90
▷ **Process Orphaning**	
▷ **Rapid-Fail Protection**	
◢ **Recycling**	
Disable Overlapped Recycle	False
Disable Recycling for Configuration Change:	False
▷ Generate Recycle Event Log Entry	
Private Memory Limit (KB)	0
Regular Time Interval (minutes)	**0**
Request Limit	0
▷ Specific Times	**TimeSpan[] Array**
Virtual Memory Limit (KB)	0

Idle Time-out (minutes)

[idleTimeout] Amount of time (in minutes) a worker process will remain idle before it shuts down. A worker process is idle if it is not processing requests and no new requests are received.

Configuring Rule Compilation to use a Background Thread

When using irCatalog, the rule engine can be configured to check for latest rules and compile rules in a background thread. The background thread compilation can help ease threading contention for applications that use multiple rule applications.

The background thread compilation feature is only available for applications that are integrated with irCatalog. The background thread is enabled by passing a Boolean flag into the constructor of the CatalogRuleApplicationReference. See the code sample below:

```
// Passing true for the last argument in the constructor will enable background
compilation
var ruleApp
= new CatalogRuleApplicationReference("MyBackgroundCompiledRuleApp", true);
```

5.5.3 Managing the Rule Application Cache

Rule Application Cache Characteristics and Significance

When a rule application is loaded for first time execution, the rule application is compiled and then cached in the AppDomain. Subsequent requests for the rule application will pull from the AppDomain cache, providing faster access to the rule application. The compile and caching process is sometimes referred to as the cold start.

Note: When calling the rule engine as a service (either with SDK, proxy objects or a custom service wrapper), the "client" will be the service itself.

Additional Information

- The FileSystemRuleApplicationReference, CatalogRuleApplicationReference and InMemoryRuleApplicationReference all leverage caching.
- Each rule application type will check to see if there have been changes made to the rule application (or if there is a new revision of the rule application in the case of the CatalogRuleApplicationReference) and will recompile and cache the rule application if there are changes.
- By default, 5 rule applications will be stored in the cache. To modify the the cache size, see Compiled Application Cache Depth
- Any time the AppDomain shuts down (e.g. IISReset or AppPool settings such as Recycling or Performance settings), the cache is cleared.
- Methods are available on the SDK to support advanced scenarios that require custom management of the rule application cache. Please see the SDK file help sections Managing the Rule Application Cache and Controlling Compilation and Cache Retention for more details.

Catalog Specific Caching Behavior

- When a rule application is pulled from the Catalog, it is cached on the client, not the Catalog Service.
- When a rule engine request executes and the rule application is not in the cache, it is pulled from the Catalog. It is then compiled and saved in the cache for a given "refresh interval".
- If the refresh interval has expired, a lightweight poll to the Catalog is performed when the next RuleSession is created to determine whether a newer revision of the rules exists. If a newer revision does not exist, the cached Rule Application is reused.
- If the Rule Application is in the cache, then the cached version is always reused until the refresh interval expires. The default refresh interval is 30 seconds. The refresh interval can be overridden by calling the SetRefresh method on the CatalogRuleApplicationReference. See the example below:

```
// Instruct the engine to work with the latest version of the rule application
var ruleApp = new CatalogRuleApplicationReference("MyRuleApp");

// Override the default refresh interval to check for latest rule version from
30 seconds to 10 minutes
ruleApp.SetRefresh(TimeSpan.FromMinutes(10));
```

- To prevent the main thread from blocking while the Catalog is polled and the latest revision of the Rule Application is compiled, the poll and compile may be performed on a background thread while the main thread continues to use the previously cached revision of the Rule Application:

```
// Instruct the engine to work with the latest version of the rule application,
but check for a new revision on a background thread after the refresh interval
has expired
var ruleApp = new CatalogRuleApplicationReference("MyRuleApp", true);
```

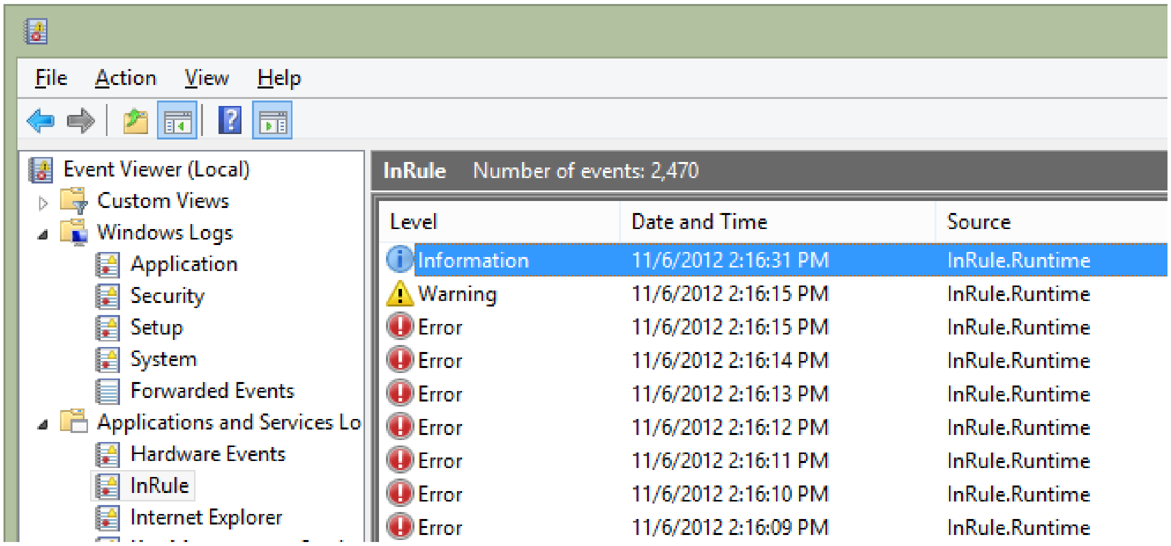
5.5.4 Event Log

Windows Events and File Logging

During rule execution, InRule will attempt to log events to either the Windows Event Log or a file-based logger. The amount and granularity of the event logging is controlled by settings exposed in the .NET configuration file for the process that is hosting InRule.

The InRule Configuration Wizard or the xcopy License Activation Tool can be used to register a Windows Event Log category on a given machine. If this category is available, then InRule events will be logged there. If the InRule event log category is not available, then events will be logged to the Windows Application log.

The InRule Category in the Windows Event Log

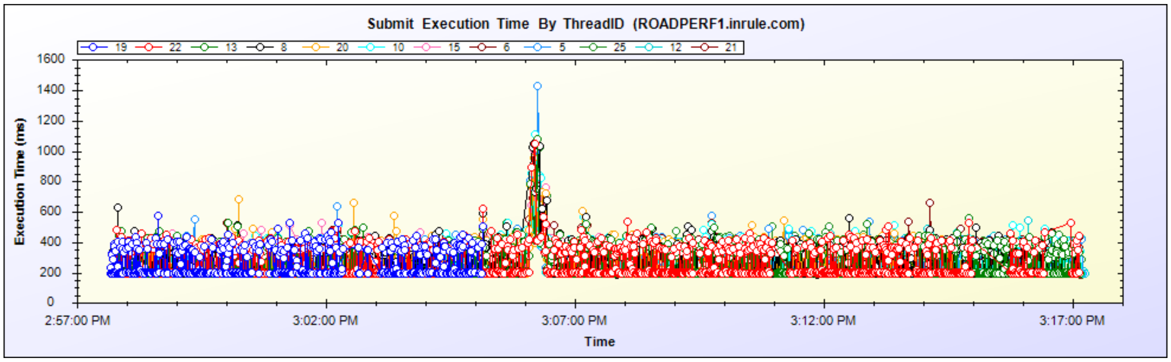


Level	Date and Time	Source
Information	11/6/2012 2:16:31 PM	InRule.Runtime
Warning	11/6/2012 2:16:15 PM	InRule.Runtime
Error	11/6/2012 2:16:15 PM	InRule.Runtime
Error	11/6/2012 2:16:14 PM	InRule.Runtime
Error	11/6/2012 2:16:13 PM	InRule.Runtime
Error	11/6/2012 2:16:12 PM	InRule.Runtime
Error	11/6/2012 2:16:11 PM	InRule.Runtime
Error	11/6/2012 2:16:10 PM	InRule.Runtime
Error	11/6/2012 2:16:09 PM	InRule.Runtime

As a best practice, the InRule.Logging section should be included in the *.config file of any process that is consuming InRule. If this section is defined, then logging levels can be toggled quickly by adjusting the value of the "level" attribute in the config file. For more information regarding configuration of InRule event logging, please see the SDK help file page InRule Logging Config File Settings. Note that if the log level is set to "Info", then the rule engine will log a light-weight summary event each time the rule engine is invoked. These summary events contain information about threading, execution times, and memory use. Some InRule implementations benefit from "info" level logging even in production, as these summary events can be reviewed to detect trends in performance and usage.

The InRule ROAD Services team has in-house tools available that can parse "info"-level log events. Below is a sample graph depicting execution times for concurrent request execution across many threads. The info level log approach can help identify and diagnose anomalies such as the unexpected spike shown in the sample.

A Sample Graph of "info"-Level Log Data-- Helping to Identify an Anomaly



.

5.5.5 Rule Engine Execution Log

Execution Log

The Execution Log provides the ability to get different types of feedback from the rules engine depending on the needs of the application. The EngineLogOptions class governs what type of information will be collected during execution and made available through the SDK. For performance sake, the default setting will not capture any information, providing the fastest possible execution. Developers can then opt into the settings that they require. The available settings are listed below along with what each setting is used for.

Option	Description
None	Default. Disables InRule.Runtime.RuleExecutionLog messages, value changes, statistics and InRule.Runtime.Tracing.IExecutionTrace generation
Execution	Enables text feedback messages and rule changes on the InRule.Runtime.RuleExecutionLog. This includes RuleSet, Rule and Action execution messages.
StateChanges	Enables state changes on the InRule.Runtime.RuleExecutionLog, including calculations.
SummaryStatisics	Enables summary statistics in InRule.Runtime.RuleExecutionLog and InRule.Runtime.RuleSession. Required for summary level information for Performance Statistics report.
DetailStatistics	Enables detailed statistics in InRule.Runtime.RuleExecutionLog. This enables RuleSet, Rule and Action timings. Required for detail level information for Performance Statistics report.
RuleTrace	Enables InRule.Runtime.Tracing.IExecutionTrace generation. This is accessible from InRule.Runtime.RuleExecutionLog.GetExecutionTrace().

	Note: While rule tracing is extremely useful for debugging rule execution, it degrades performance significantly and is not intended to be used in production environments.
RuleValues	Enables rule values to be captured so that they can be accessed using the SDK. Adding this flag increases memory consumption and reduces performance. When this option is off, attempting to access certain rule values or execution counts for a child element from the SDK will result in an IntegrationException, which will be one of the following error codes: • SDKRuleValuesCanOnlyBeAccessedWhenStoringRuleValues • SDKRuleExecutionCountCanOnlyBeAccessedWhenStoringRuleValues

The EngineLogOptions is a flag enumeration, allowing multiple options to be specified for a single call as shown below:

```
    ruleSession.Settings.LogOptions = EngineLogOptions.Execution |
EngineLogOptions.Changes;
```

Availability

The RuleExecutionLog is available as the return value from both the RuleSession.ApplyRules() and Entity.ExecuteRuleSet() methods. It is also available from the RuleSession.LastRuleExecutionLog property.

Please see the SDK help file section Retrieving and Processing the RuleExecutionLog for more details.

5.5.6 Hosting irCatalog on IIS vs Windows Services

Background

During setup and installation of the irServer components, irCatalog, and irServer Rule Execution Service, the installer can choose to host the component services on IIS or as a Windows Service.

Managed Windows Services

This option registers the WCF service as a managed Windows Service. Key benefits and limitations of managed windows services are as follows:

- Lifetime of the service process is controlled by the operating system and is not message activated.
- Most versions of Windows support managed Windows Services

IIS

IIS is recommended as the preferred approach for hosting InRule in service applications. Key benefits and IIS are noted below:

- Process recycling and health monitoring
- Message-based activation
- Built-in request throttling
- Built-in thread pooling and request queuing

As detailed in the InRule Installation Help File, InRule generally recommends IIS service types for irCatalog and irServer Rule Execution Service because of the built-in failover, queuing, administration, deployment, and scalability capabilities, as well as additional configuration options.

Part VI

Source Code Examples

6 Source Code Examples

For most rule engine implementations, it is a common scenario to integrate with one or more end business applications or processes. InRule provides an extensive developer API (irSDK) to incorporate the decisions and actions authored in the rule application to drive the processes within an end business application.

These examples describe the methods and techniques to perform many of the common functions used by a typical rule-enabled application. The following topics are covered in this section:

Runtime API Examples

The InRule Runtime API contains all of the objects and methods used to load data, execute rules, process results, and work with rule application metadata.

Authoring API Examples

The InRule Authoring API (also referred to as the Repository SDK) spans two functional areas: developing against the RuleApplicationDef model and embedding authoring controls.

Catalog API Examples

The InRule Catalog API demonstrates how to retrieve and work with rule applications that are stored in the InRule Catalog.

6.1 Runtime API Examples

The InRule Runtime API contains all of the objects and methods used to load data, execute rules, process results, and work with rule application metadata.

Calling the Rules Engine - In Process

- Basic Example of Calling the Rule Engine
- Opening a Rule Application for Execution
- Creating a RuleSession
- Creating a RuleSession with Cache Retention
- Creating Entities
- Alternate ways to Load State for Entities
- Retrieving Entity State
- Working with RuleSessionState

Calling the Rules Engine as a Service

- Sychronously calling the REST Service
- Asychronously calling the REST Service with JSON
- Asychronously calling the REST Service with XML
- Calling the SOAP EndPoint using a Service Reference

Retrieving and Setting Fields and Entities

- Retrieving an Entity from the RuleSession
- Retrieving Fields
- Setting Fields

Working with Collections

- Looping through a Collection
- Adding a Member to a Collection
- Resolving Field Types at Runtime

Executing Rules

- Applying Rules
- Applying Rules with an Activation Model
- Executing Decisions
- Executing an Independent Rule Set or Rule Flow
- Checking For Notifications & Validations
- Handling Exceptions
- Runtime Settings
- Retrieving and Processing the RuleExecutionLog
- Retrieve the Performance Statistics Report
- Retrieve the Performance Log

Working with the Rule Application Cache

- Adding Items into the Cache
- Managing the Cache
- Controlling Compilation and Cache Retention
- Iterating Items in the Cache

Working with Rule Application Metadata

- Using the RuleApplicationDef Object
- Retrieving Definition Objects at Runtime
- Using Element Metadata
- Working with Attributes
- Working with Value Lists

Other

- Launching irVerify or the StateViewer From Code
- Overriding EndPoints at Runtime
- Overriding DataElements at Runtime
- Overriding Culture Settings at Runtime
- Rule Tracing Input and Output Using RuleSession
- Working with the Trace Viewer Through irSDK
- Executing a Simple Test Suite

6.1.1 Calling the Rules Engine - In Process

Calling the Rules Engine - In Process

- Basic Example of Calling the Rule Engine
- Opening a Rule Application for Execution
- Creating a RuleSession
- Creating a RuleSession with Cache Retention

- Creating Entities
- Alternate ways to load state for Entities
- Retrieving Entity State
- Working with RuleSessionState

6.1.1.1 Basic Example of Calling the Rule Engine

Prerequisites: None
Namespaces: InRule.Runtime, InRule.Common.Exceptions
Classes: FileSystemRuleApplicationReference, RuleSession, Entity, RuleException
See Also: Retrieving a Rule Application, Creating a RuleSession, Creating Entities, Applying Rules, Handling Exceptions
References: InRule.Runtime.dll, InRule.Common.dll, InRule.Repository.dll

The following basic example demonstrates calling the rules engine to calculate the area of a rectangle.

```
// Get the ruleapp from a file.
RuleApplicationReference ruleAppRef = new FileSystemRuleApplicationReference(@"C:
\RuleApps\Rectangle.ruleapp");

// Create a session to manage the rule engine request and response.
// The RuleSession should be disposed after use. This can be done with a "using"
statement or by explicitly calling Dispose
using (RuleSession session = new RuleSession(ruleAppRef))
{
    try
    {
           // Create the "Rectangle" Entity defined in the rule application schema
and set the values using xml.
           Entity rectangle = session.CreateEntity("Rectangle",
@"<Rectangle><Height>2</Height><Width>3</Width></Rectangle>");

           // Apply the rules to calculate the area of the rectangle.
           session.ApplyRules();

           // Get the state and results as XML
           string resultsXml = rectangle.GetXml();

    }
    // catch RuleException, base class for all InRule Exceptions
    catch (RuleException ex)
    {
           Console.WriteLine(ex.ToString());
    }
}
```

Note: There are specific InRule exceptions that can be caught, including Compiler, Runtime and several other exceptions. See Handling Exceptions for samples. See RuleException Class for a complete list of InRule exceptions.

6.1.1.2 Opening a Rule Application for Execution

Prerequisites: None
Namespaces: InRule.Runtime
Classes: FileSystemRuleApplicationReference, InMemoryRuleApplicationReference, CatalogRuleApplicationReference

InRule provides the ability to execute rules from the file system, catalog, or a rule application created in memory.

FileSystemRuleApplicationReference

```
// Get the ruleapp from the file system.
RuleApplicationReference ruleAppRef = new FileSystemRuleApplicationReference("C:\
\RuleApps\\Invoice.ruleapp");
```

CatalogRuleApplicationReference

```
// The URI for the catalog service.
string Uri = "http://localhost/InRuleCatalogService/Service.svc";

// Get the ruleapp from the repository using the URI, Username, Password, and Rule
Application name.
CatalogRuleApplicationReference ruleAppRef = new
CatalogRuleApplicationReference(Uri, "CustomerInvoice", "Admin", "password");

// If you are using active directory,  you only need the Uri and application name
ruleAppRef = new CatalogRuleApplicationReference(Uri, "CustomerInvoice");

// you can also specify a version e.g 2
ruleAppRef = new CatalogRuleApplicationReference(Uri, "CustomerInvoice", "Admin",
"password",2);

// Using a label for example PRODUCTION
ruleAppRef = new CatalogRuleApplicationReference(Uri, "CustomerInvoice", "Admin",
"password", "PRODUCTION");

// Using background version polling and recompilation after the refresh interval
has expired
ruleAppRef = new CatalogRuleApplicationReference(Uri, "CustomerInvoice", "Admin",
"password", "PRODUCTION", true);
```

InMemoryRuleApplicationReference

```
// Open an InMemoryRuleApplicationReference  using the RuleApplicationDef object
// Used when you have a rule RuleApplicationDef object available in memory,
// for example when dynamically generating rule applications via the Repository
SDK.
RuleApplicationReference ruleAppRef = new InMemoryRuleApplicationReference
(ruleApplicationDef);
```

6.1.1.3 Creating a RuleSession

Prerequisites: A valid RuleApplicationReference
Namespaces: InRule.Runtime
Classes: RuleSession, RuleApplicationReference

See Also: Retrieving a Rule Application, Creating a RuleSession with Cache Retention

The rule session object object that manages all of the rules engine request directives and execution results. The rule session is used to load state, execute rules, and retrieve notifications, validations, and the execution log.

```
// Create a session to the rules engine, passing in a RuleApplicationReference
RuleSession ruleSession = new RuleSession(ruleAppRef);

// Use of cache retention
RuleSession ruleSession = new RuleSession(ruleAppRef, CacheRetention.AlwaysRetain);


// Using  Rule Application Definitions  reference to create a rule session
RuleSession ruleSession = new RuleSession(ruleAppDef);

// Using Rule Application Definition that has been saved to file to create a rule
session
RuleSession ruleSession = new RuleSession(@"C:\temp\ruleAppDef");
```

6.1.1.4 Creating a RuleSession with Cache Retention

Prerequisites: A valid RuleApplicationReference
Namespaces: InRule.Runtime
Classes: RuleSession, RuleApplicationReference, CacheRetention
See Also: Retrieving a Rule Application, Adding items into the Cache, Controlling Compilation and Cache Retention, Iterating items in the cache , Working with the Rule Application Cache

The rule session object object that manages all of the rules engine request directives and execution results. The rule session is used to load state, execute rules, and retrieve notifications, validations, and the execution log.

Cache retention controls how a session is stored in the cache

```
// Create a rule session with a rule application reference and
// specifying the cache retention
RuleSession ruleSession = new RuleSession(ruleAppRef, CacheRetention.AlwaysRetain);


// Create a rule session with a rule application definition reference and
// specifying the cache retention
RuleSession ruleSession = new RuleSession(ruleAppDef, CacheRetention.AlwaysRetain);

// Using Rule Application Definition that has been saved to file to create
// a rule session and specifying the cache retention
RuleSession ruleSession = new RuleSession(@"C:\temp\ruleAppDef",
CacheRetention.AlwaysRetain);

// Using the cache retention weight to control how an rule application will be kept
in the cache
// Higher values are more likely to be retained; Lower valies are less likely to be
retained
```

```
// and therefore expired from the cache. 1000 is the default weight.
RuleSession ruleSession = new RuleSession(ruleAppRef,
CacheRetention.FromWeight(5000));
```

6.1.1.5 Creating Entities

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime
Classes: Entity, RuleSession
See Also: Retrieving a Rule Application, Creating a RuleSession

For typical entity models, there is a top level entity that provides access to all fields, sub-entities, and collections defined in the rule application. The alternative independent entity model approach requires the use of Independent Rulesets.

Create blank Entity

```
//create a blank mortgage entity
Entity mortgageEntity = ruleSession.CreateEntity("Mortgage");
```

Create Entities with XML

```
// Create a mortgage entity with XML
Entity mortgageEntity = ruleSession.CreateEntity("Mortgage",@"<Mortgage>
         <LoanInfo><PropertyId>1</PropertyId><Principal>500000</Principal>
         <APR>6.875</APR><TermInYears>30</TermInYears></LoanInfo><PaymentSummary/
></Mortgage>");
```

Create the Entity model from a business object model

```
// Create a mortgage business object
Mortgage mortgage = new Mortgage();

LoanInfo loanInfo = new LoanInfo();
loanInfo.APR = 6.875m;
loanInfo.PropertyId = 1;
loanInfo.Principal = 500000;
loanInfo.TermInYears = 30;

mortgage.LoanInfo = loanInfo;
mortgage.PaymentSummary = new PaymentSummary();

Entity mortgageEntity = ruleSession.CreateEntity("Mortgage", mortgage);
```

Note: See also: Setting Fields examples

Note: In order to use object-based state, the rule application must first be bound to a .NET assembly schema. See Binding to a .NET Assembly Schema in the main InRule Help file.

6.1.1.6 Alternate ways to Load State for Entities

Prerequisites: A valid Entity
Namespaces: InRule.Runtime
Classes: Entity
See Also: Retrieving a Rule Application, Creating a RuleSession, Creating Entities

Loading state from a file

```
// Load the state by passing in the path & filename.
mortgageEntity.LoadXml("Mortgage.xml");
```

Loading state from an XML string

```
// Load state from an XML string.
mortgageEntity.ParseXml("<Mortgage><LoanInfo><Principal>500000</Principal><APR>7</
APR><TermInYears>30</TermInYears></LoanInfo><PaymentSummary /></Mortgage>");
```

Manually wiring up an Entity

```
//create ruleapp entities
Entity mortgageEntity = ruleSession.CreateEntity("Mortgage");
Entity loanInfo = ruleSession.CreateEntity("LoanInfo");
Entity paymentSummary = ruleSession.CreateEntity("PaymentSummary");

//set loanInfo values
loanInfo.Fields["Principal"].Value = 250000;
loanInfo.Fields["APR"].Value = 5.25;
loanInfo.Fields["TermInYears"].Value = 30;

//associate loanInfo entity to an Mortgage entity field
mortgageEntity.Fields["LoanInfo"].Value = loanInfo;

//associate paymentSummary entity to Mortgage entity field
mortgageEntity.Fields["PaymentSummary"].Value = paymentSummary;
```

Note: In order to use object-based state, the rule application must first be bound to a .NET assembly schema. See Binding to a .NET Assembly Schema in the main InRule Help file.

6.1.1.7 Retrieving Entity State

Prerequisites: A valid Entity
Namespaces: InRule.Runtime
Classes: Entity
See Also: Retrieving a Rule Application, Creating a RuleSession, Creating Entities

Saving the state to the file system

```
// Save the state by passing in the path & filename.
mortgageEntity.SaveXml("MortgageOutput.xml");
```

Retrieving entity state as object

```
// get a reference to the object model - provided an object was used to
// create (or load) the entity
Mortgage mortgageObject = mortgageEntity.BoundValue as Mortgage;
```

Retrieving entity state as XML

```
// Get the state as XML
string stateXml = mortgageEntity.GetXml();
```

6.1.1.8 Working with RuleSession State

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime
Classes: Entity, RuleSession
See Also: Retrieving a Rule Application, Creating a RuleSession, Creating Entities

The following demonstrates how to serialize RuleSession state, also known as a Test Scenario, to the file system and then subsequently load the RuleSession state back into memory. The persisted file on the file system can be loaded directly in irVerify by selecting File --> Load Test Scenario and selecting the file.

Saving RuleSession State to the File System

```
// Save RuleSession state
ruleSession.SaveState(@"c:\work\state.testscenario");
```

Loading RuleSession State from the File System

```
// Load RuleSession state
ruleSession.LoadState(@"c:\work\state.testscenario");
```

6.1.2 Calling the Rules Engine as a Service

Calling the Rules Engine as a Service

- Sychronously calling the REST Service
- Asychronously calling the REST Service with JSON
- Asychronously calling the REST Service with XML
- Calling the SOAP EndPoint using a Service Reference

See Also: irServer - Rule Execution Service

6.1.2.1 Synchronously calling the REST Service

See Also: irServer - Rule Execution Service

The following example demonstrates calling the irServer - Rule Execution Service REST interface synchronously with the legacy WebRequest class.

```
// The name of the ruleApp in the Catalog
string ruleApp = "MortgageCalculator";

// The address of the Rule Execution Service
string ruleExecutionServiceURI = "http://localhost/InRuleRuleEngineService_v5.0.16/
HttpService.svc";
```

```
// The name of the Entity
string entityName = "Mortgage";

// The name of the RuleSet to execute, or use 'ApplyRules' if found to be null.
string ruleSetName = "PaymentSummaryRules";

// The entity state as a C# dynamic type
dynamic entityState = new
{
    LoanInfo = new
    {
        Principal = 53000m, APR = 7.625f, TermInYears = 15
    }
};

 // We use Newtonsoft's Json.Net library to convert this dynamic C# object to a JSON string
 string entityStateJsonInput = JsonConvert.SerializeObject(entityState);

 // New create the full request, inserting the JSON string that represents EntityState.
 dynamic requestData = new
 {
     RuleApp = new
     {
         RepositoryRuleAppRevisionSpec = new
         {

             RuleApplicationName = ruleApp
         }
     },
     EntityName = entityName,
     EntityState = entityStateJsonInput,
     RuleSetName = ruleSetName
 };

 string requestDataString = JsonConvert.SerializeObject(requestData);

 //If no ruleSetName exists, execute Auto rules, otherwise execute explicit rule.
 string postUri;
 if (string.IsNullOrEmpty(ruleSetName))
 { postUri = ruleExecutionServiceURI + "/ApplyRules"; }
 else
 { postUri = ruleExecutionServiceURI + "/ExecuteRuleSet"; }

 // Encode our string into an array of bytes, needed by WebRequest.
 UTF8Encoding encoding = new UTF8Encoding();
 byte[] bytes = encoding.GetBytes(requestDataString);

 // Create and prepare the request
 WebRequest webRequest = WebRequest.Create(postUri);

 // ApplyRules only works as a POST, not a GET
 webRequest.Method = "POST";

 // Our EntityState is encoded as an JSON document, not XML.
 webRequest.ContentType = "application/json";

 // WebRequest is old, and can't count by itself.
 webRequest.ContentLength = bytes.Length;

 string httpResponse;

 // Create our requestStream by opening the request, make sure it gets disposed when done.
 using (var requestStream = webRequest.GetRequestStream())
 {
     // send the data
     requestStream.Write(bytes, 0, bytes.Length);
 }


 // In case it has not already been mentioned this is a rather primitive way of
 // making an HTTP based request.  We recommend you read the HttpClient based code samples
 // elsewhere in this document, as that Asynchronous approach is a better way to go.
 try
 {
     HttpWebResponse response = webRequest.GetResponse() as HttpWebResponse;
     using (var stream = response.GetResponseStream())
     {
         var reader = new StreamReader(stream, Encoding.UTF8);
         httpResponse = reader.ReadToEnd();
     }

 }
 catch (WebException ex)
 {
     using (var stream = ex.Response.GetResponseStream())
     {
         var reader = new StreamReader(stream, Encoding.UTF8);
         httpResponse = reader.ReadToEnd();
         throw new Exception(httpResponse, ex);
     }
 }

dynamic resultJson = JsonConvert.DeserializeObject<dynamic>(httpResponse);
string entityJsonString = resultJson.EntityState;
dynamic entityJson = JsonConvert.DeserializeObject<dynamic>(entityJsonString);
string paymentSummaryJsonString = JsonConvert.SerializeObject(entityJson.PaymentSummary);
```

6.1.2.2 Asynchronously calling the REST Service with JSON

See Also: irServer - Rule Execution Service

The following example demonstrates calling the irServer - Rule Execution Service REST interface with a JSON based Entity State.

```
static void Main(string[] args)
{

    // The name of the ruleApp in the Catalog
    string ruleApp = "MortgageCalculator";

    // The address of the Rule Execution Service
    string ruleExecutionServiceURI = "http://localhost/InRuleRuleEngineService_v5.0.16/
HttpService.svc";

    // The name of the Entity
    string entityName = "Mortgage";

    // The name of the RuleSet to execute, or use 'ApplyRules' if found to be null.
    string ruleSetName = "PaymentSummaryRules";

    // The entity state as a C# dynamic type
    dynamic entityState = new
    {
        LoanInfo = new
        {
            Principal = 53000m, APR = 7.625f, TermInYears = 15
        }
    };

    // A console application's main() entry point cannot be declared as async, so here
    // we are creating an Async task to execute the Async method, then we will wait on
    // the result.
    Task<string> httpTask = AsyncRestJsonCatalog(ruleExecutionServiceURI, ruleApp,
                                            entityName, entityState, ruleSetName);

    // Calling the Result property of a Task object will block this thread until
    // the Task has had a chance to complete in it's worker thread.
    string httpTaskResult = httpTask.Result;

    // After this call you can inspect the raw result (as JSON) in resultJson, but note
    // that the EntityState will be serialized Json and must be decoded.
    dynamic resultJson = JsonConvert.DeserializeObject<dynamic>(httpTaskResult);
    string entityJsonString = resultJson.EntityState;

    // Decode the serialized entity Json into a dynamic entity.
    dynamic entity = JsonConvert.DeserializeObject<dynamic>(entityJsonString);

}

private async static Task<string> AsyncRestJsonCatalog(string ruleExecutionServiceURI,
string ruleApp,
                     string ruleApplicationEntityName, dynamic entityStateObjectInput,
string ruleSetName)
{

    // We use Newtonsoft's Json.Net library to convert this dynamic C# object to a JSON string
    string entityStateJsonInput = JsonConvert.SerializeObject(entityStateObjectInput);

    // New create the full request, inserting the JSON string that represents EntityState.
    dynamic requestData = new
    {
        RuleApp = new
        {
            RepositoryRuleAppRevisionSpec = new
            {

                RuleApplicationName = ruleApp
            }
        },
        EntityName = ruleApplicationEntityName,
        EntityState = entityStateJsonInput,
        RuleSetName = ruleSetName
    };

    string requestDataString = JsonConvert.SerializeObject(requestData);

    //If no ruleSetName exists, execute Auto rules, otherwise execute explicit rule.
    string postUri;
    if (string.IsNullOrEmpty(ruleSetName))
    { postUri = ruleExecutionServiceURI + "/ApplyRules"; }
    else
    { postUri = ruleExecutionServiceURI + "/ExecuteRuleSet"; }

    // Our EntityState is encoded as JSON document, not XML.
    string mediaType = "application/json";

    // Async call to HttpClient with an HttpContent and wait for result.
    HttpContent content = new StringContent(requestDataString, Encoding.UTF8, mediaType);
    HttpClient client = new HttpClient();
    HttpResponseMessage response = await client.PostAsync(postUri, content);

    // Async convert the HttpResponseMessage's content to a string.
    string responsestring = await response.Content.ReadAsStringAsync();

    // We're done, return result.
    return responsestring;

}
```

6.1.2.3 Asynchronously calling the REST Service with XML

See Also: irServer - Rule Execution Service

The following example demonstrates calling the irServer - Rule Execution Service REST interface with an XML based Entity State.

```
static void Main(string[] args)
{

   // The name of the ruleApp in the Catalog
   string ruleApp = "MortgageCalculator";

   // The address of the Rule Execution Service
   string ruleExecutionServiceURI = "http://localhost/InRuleRuleEngineService/HttpService.svc";

   // The Entity Name:
   string entityName = "Mortgage";

   // The name of the RuleSet to execute, or use 'ApplyRules' if found to be null.
   string ruleSetName = "PaymentSummaryRules";

   // The Entity State as XML text.
   string entityStateXMLInput = @"<?xml version='1.0' encoding='utf-8'?>
<Mortgage xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>
    <LoanInfo>
        <Principal>53000</Principal>
        <APR>7.625</APR>
        <TermInYears>15</TermInYears>
    </LoanInfo>
    <PaymentSummary/>
</Mortgage>";

   // A console application's main() entry point cannot be declared as async, so here
   // we are creating an Async task to execute the Async method, then we will wait on
   // the result.
   Task<string> httpTask = AsyncRestXmlCatalog(ruleExecutionServiceURI, ruleApp, entityName,
entityStateXMLInput, ruleSetName);

   // Calling the Result property of a Task object will block this thread until
   // the Task has had a chance to complete in it's worker thread.
   string httpTaskResult = httpTask.Result;

   Console.WriteLine(httpTaskResult);

   // Now let's extract the <EntityState> element, which has been encoded.
   // The Value property of the XElement will automatically decode the contents
   // and expose it as a string, which lets us look at the inner XML document "EntityState".
   // It's worth observing that while the input was encoded as UTF-8, this result is in
   // UTF-16.
   XDocument xmldoc = XDocument.Parse(httpTaskResult);
   string xmlEntityStateOutput =
       (from n in xmldoc.Descendants()
        where n.Name.LocalName == "EntityState"
        select n.Value).FirstOrDefault();

   // Wonderful!  Now lets take that concept further and extract just the
   // <PaymentSummary> node from the document, which gives us our computed "answers".
   XDocument xmlentitystatedoc = XDocument.Parse(xmlEntityStateOutput);
   string paymentSummaryXml = (from n in xmlentitystatedoc.Descendants()
                               where n.Name.LocalName == "PaymentSummary"
                               select n.ToString()).FirstOrDefault();



   }

private async static Task<string> AsyncRestXmlCatalog(string ruleExecutionServiceURI,
                                                         string ruleApp,
                                                         string entityName,
                                                         string entityStateXMLInput,
                                                         string ruleSetName)
{

   // We must encode the Entity State XML into a valid XML string so that
   // it can be wrapped with the outer XML request document.
   string entityStateXMLEncoded = SecurityElement.Escape(entityStateXMLInput);

   string rootNodeName;
   if (string.IsNullOrEmpty(ruleSetName))
   { rootNodeName = "ApplyRulesRequest"; }
   else
   { rootNodeName = "ExecuteRuleSetRequest"; }


   // Now we generate the outer XML request document.
   string requestXML = $@"<?xml version='1.0' encoding='utf-8'?>
<{rootNodeName} xmlns='http://www.inrule.com/XmlSchema/Schema'>
    <RuleApp>
        <RepositoryRuleAppRevisionSpec>
            <RuleApplicationName>{ruleApp}</RuleApplicationName>
        </RepositoryRuleAppRevisionSpec>
    </RuleApp>
    <EntityState>{entityStateXMLEncoded}</EntityState>
    <EntityName>{entityName}</EntityName>
    <RuleSetName>{ruleSetName}</RuleSetName>
</{rootNodeName}>";

   //If no ruleSetName exists, execute Auto rules, otherwise execute explicit rule.
   string postUri;
   if (string.IsNullOrEmpty(ruleSetName))
   { postUri = ruleExecutionServiceURI + "/ApplyRules"; }
   else
   { postUri = ruleExecutionServiceURI + "/ExecuteRuleSet"; }

   // Our EntityState is encoded as an XML document, not JSON.
```

```
    string mediaType = "application/xml";

    // Async call to HttpClient with an HttpContent and wait for result.
    HttpContent content = new StringContent(requestXML, Encoding.UTF8, mediaType);
    HttpClient client = new HttpClient();
    HttpResponseMessage response = await client.PostAsync(postUri, content);

    // Async convert the HttpResponseMessage's content to a string.
    string responseXML = await response.Content.ReadAsStringAsync();

    // We're done, return result.
    return responseXML;
}
```

6.1.2.4 Calling the SOAP EndPoint using a Service Reference

Prerequisites: A valid Service Reference
See Also: irServer - Rule Execution Service

The following sample represents how to call irServer Rule Execution Service SOAP EndPoint using a service reference:

```
using (RuleEngineServiceClient proxy = new RuleEngineServiceClient())
{
    try
    {
          // Get RuleApp as defined in the config (RepositoryRuleApp or
FileSystemRuleApp)
          RepositoryRuleApp rules = new RepositoryRuleApp();
          rules.RepositoryServiceUri = "http://server/InRuleCatalogService/
Service.svc";
          RepositoryRuleAppRevisionSpec spec = new
RepositoryRuleAppRevisionSpec();
          spec.RuleApplicationName = "MortgageCalculator";
          rules.RepositoryRuleAppRevisionSpec = spec;
          rules.UserName = "Admin";
          rules.Password = "password";

          // Create new ApplyRulesRequest
          ApplyRulesRequest request = new ApplyRulesRequest();
          request.RuleApp = rules;
          request.EntityName = "Mortgage";
          request.RuleEngineServiceOutputTypes = new
RuleEngineServiceOutputTypes();
          request.RuleEngineServiceOutputTypes.ActiveNotifications = true;
          request.RuleEngineServiceOutputTypes.ActiveValidations = true;
          request.RuleEngineServiceOutputTypes.EntityState = true;

          // Load state XML
          Console.WriteLine("- Loading XML state for 'Invoice'...");
          request.EntityState = "<Mortgage><LoanInfo><Principal>500000</
Principal><APR>6.875</APR>< TermInYears > 30 </TermInYears ></LoanInfo
><PaymentSummary/ ></Mortgage > ";
          Console.WriteLine("Input State:");
          Console.WriteLine(request.EntityState);
          Console.WriteLine("");

          // Submit Request
          Console.WriteLine("- Calling ApplyRules() from RuleEngineService...");
          RuleEngineServiceResponse response =
proxy.ExecuteRuleEngineRequest(request);

          Console.WriteLine("Active Notifications:");
          foreach (Notification notification in response.ActiveNotifications)
          {
                 Console.WriteLine(notification.NotificationType + ": " +
notification.Message);
          }
          Console.WriteLine("");

          Console.WriteLine("Active Validations:");
          foreach (Validation validation in response.ActiveValidations)
          {
                 Console.WriteLine(validation.InvalidMessageText);
          }
          Console.WriteLine("");

          Console.WriteLine("Output State:");
          // Note: XML formatting not maintained in response
          Console.WriteLine(response.EntityState);
          Console.WriteLine("");
   }
   catch (Exception ex)
   {
          Console.WriteLine("Unknown exception occurred during RuleEngineService
request: " + ex.ToString());
   }
}

Console.WriteLine("[END ServiceReferenceConsumer Sample]");
```

6.1.3 Retrieving and Setting Fields and Entities

Retrieving and Setting Fields and Entities

- Retrieving an Entity from the RuleSession
- Retrieving Fields
- Setting Fields

6.1.3.1 Retrieving an Entity from the RuleSession

Prerequisites: A valid RuleSession, A valid Entity
Namespaces: InRule.Runtime
Classes: RuleSession, Entity
See Also: Retrieving a Rule Application, Creating a RuleSession, Creating Entities

To retrieve an Entity from the RuleSession, use the GetEntity() method. The input to GetEntity() may contain either the Element ID or the Instance ID, if one has been assigned.

If no Entity is found, GetEntity() returns null.

Obtain the Element ID

```
// Obtain the Element ID to lookup the Entity from the RuleSession at a later point
string elementId = ruleSession.CreateEntity("Invoice").ElementId;
```

Retrieve the Entity using the Element ID

```
// Get the Entity from the RuleSession using the Element ID
Entity entity = ruleSession.GetEntity(elementId);
if (entity == null)
{
    // Entity was not found
}
```

Retrieve the Entity using an Instance ID

```
// Get the Entity from the RuleSession using the Element ID
Entity entity = ruleSession.GetEntity("Invoice175941");
if (entity == null)
{
    // Entity was not found
}
```

6.1.3.2 Retrieving Fields

Prerequisites: A valid Entity
Namespaces: InRule.Runtime
Classes: Field, Collection, Entity
See Also: Retrieving a Rule Application, Creating a RuleSession, Creating Entities

Retrieving a field from the entity

```
//  Retrieve the CustomerID field from the Invoice entity.
Field customerId = invoiceEntity.Fields["CustomerID"];
```

Retrieving a field from a collection

```
// Retrieve the ProductID of the first LineItem in the LineItems collection.
Field productId = invoiceEntity.Collections["LineItems"][0].Fields["ProductID"];
```

Note: Collection indexing is 0-based through the SDK and 1-based for referencing from within a rule application.

Retrieving a field to set a typed variable

```
//  Retrieve the CustomerID field value from the Invoice entity.
int customerId = invoiceEntity.Fields["CustomerID"].Value.ToInt32();
```

Retrieving all fields in an Entity

```
// Show all the fields in the Entity
foreach (Field field in invoiceEntity.Fields)
{
   Console.WriteLine(string.Format("Found {0} {1}", field.Name,
field.Value.ToString()));
}
```

6.1.3.3 Setting Fields

Prerequisites: A valid Entity
Namespaces: InRule.Runtime
Classes: Field, Collection, Entity
See Also: Retrieving a Rule Application, Creating a RuleSession, Creating Entities, Retrieving Fields

Setting a field in the entity

```
// Set the CustID field to 2
invoiceEntity.Fields["CustomerID"].Value = 2;
```

Setting a field from a collection

```
// Set the ProductID field of the first LineItem in the LineItems collection.
invoiceEntity.Collections["LineItems"][0].Fields["ProductID"].Value = 6;
```

Note: Collection indexing is 0-based through the SDK and 1-based for referencing within rules.

6.1.4 Working with Collections

Working with Collections

- Looping through a Collection
- Adding a Member to a Collection
- Resolving Field Types at Runtime

6.1.4.1 Looping Through a Collection

Prerequisites: A valid Entity
Namespaces: InRule.Runtime
Classes: Field, Collection, Entity
See Also: Retrieving a Rule Application, Creating a RuleSession, Creating Entities

Entity based collections

```
// loop over all the payments collection members
foreach (EntityCollectionMember payment in mortgageEntity.Collections["Payments"])
{
   // Write out some of the entity field values
```

```
    Console.WriteLine(payment.Fields["PaymentDate"].Value.ToString());
    Console.WriteLine(payment.Fields["Amount"].Value.ToString());
    Console.WriteLine(payment.Fields["RemainingBalance"].Value.ToString());
}
```

An alternative way to loop through the entities themselves is as follows:

```
// Check if the collection of entities
if (collection is InRule.Runtime.EntityCollection)
{
    // process a collection of entities
    foreach (EntityCollectionMember member in collection)
    {
        // Get the entity the member refers to
        Entity memberEntity = member.Value.ToEntity();
        // Iterate the fields
        foreach (Field field in memberEntity.Fields)
        {
            // get the fields info
            Console.WriteLine("FieldName:" + field.Name);
            Console.WriteLine("FieldValue:" + field.Value.ToString());
        }
    }
}
```

Complex collections

```
// Loop through the members in the complex collection AdditionalCharges
foreach (CollectionMember charge in
mortgageEntity.Collections["AdditionalCharges"])
{
    // write the field values
    Console.WriteLine(charge.Fields["Name"].Value.ToString());
    Console.WriteLine(charge.Fields["Amount"].Value.ToString());
}
```

Note: Collection indexing is 0-based through the SDK and 1-based for referencing within rules

6.1.4.2 Adding a Member to a Collection

Prerequisites: A valid Entity
Namespaces: InRule.Runtime
Classes: Collection, Entity
See Also: Retrieving a Rule Application, Creating a RuleSession, Creating Entities

Adding a new member to a Collection

```
// Create top level entity
Entity invoiceEntity = ruleSession.CreateEntity("Invoice");

// Get reference to the LineItems collection
Collection lineItemCollection = invoiceEntity.Collections["LineItems"];

// Add a new line item to the collection
CollectionMember lineItem = (lineItemCollection.Add());
```

```
// Get underlying entity
Entity lineItemEntity = lineItem.Value.ToEntity();

// Update the line item fields on the newly added member
lineItemEntity.Fields["ProductID"].Value = 2;
lineItemEntity.Fields["Quantity"].Value = 10;
```

Adding an existing member to a Collection

```
// Create top level entity
Entity invoiceEntity = ruleSession.CreateEntity("Invoice");

// Get reference to the LineItems collection
Collection lineItemCollection = invoiceEntity.Collections["LineItems"];

// Create a LineItem and populate the fields
Entity lineItemEntity = ruleSession.CreateEntity("LineItem");
lineItemEntity.Fields["ProductID"].Value = 3;
lineItemEntity.Fields["Quantity"].Value = 100;

// Append the new member to the collection
lineItemCollection.Add(lineItemEntity);
```

6.1.4.3 Resolving Field Types at Runtime

Prerequisites:
Namespaces: InRule.Runtime
Classes: FileSystemRuleApplicationReference, RuleSession, Entity
See Also:
References: InRule.Runtime.dll, InRule.Common.dll

To resolve the field type, you can use the following code:

```
// Test if this field refers to an entity
if (field.GetType() == typeof(InRule.Runtime.EntityField))
{

}
```

In a similar way, you can test if a collection contains entities:

```
// Test if this is a collection of entities
if (collection.GetType() == typeof(InRule.Runtime.EntityCollection))
{
}
```

6.1.5 Executing Rules

Executing Rules

- Applying Rules
- Applying Rules with an Activation Model
- Executing Decisions
- Executing an Independent Rule Set or Rule Flow
- Checking For Notifications & Validations
- Handling Exceptions
- Runtime Settings
- Retrieving and Processing the RuleExecutionLog
- Retrieving the Performance Statistics Report
- Retrieving the Performance Log

6.1.5.1 Applying Rules

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime
Classes: RuleSession, Entity, RuleException
See Also: Retrieving a Rule Application, Creating a RuleSession, Creating Entities, Handling Exceptions
References: InRule.Runtime.dll, InRule.Common.dll

Apply rules

```
try
{
    // Apply the rules.
    session.ApplyRules();
}
catch (RuleException ex)
{
    // Handle any rule engine exceptions here.
    Console.WriteLine(ex.ToString());
}
```

Note: This will execute all Auto RuleSets, calculations, constraints and classifications.

Execute explicit rulesets and rule flows

```
// Apply the CalculatePaymentSchedule from the mortgageEntity
mortgageEntity.ExecuteRuleSet("CaculatePaymentSchedule");
```

Notes:

Rule flows are executed using the same method and behave in the same manner as explicit rulesets.
Any auto rules are also be applied when applying an explicit ruleset or rule flow

Execute explicit rulesets that accept parameters

```
// Execute the ruleset passing in a list of parameters
mortgageEntity.ExecuteRuleSet("AddCharge", "Appraisal", 400.99);
```

6.1.5.2 Applying Rules with an Activation Model

Prerequisites: A valid RuleSession and Entity
Namespaces: InRule.Runtime
Classes: RuleSession, Entity
References: InRule.Runtime.dll, InRule.Common.dll

The following examples demonstrate how to control rule execution by Activating and Deactivating RuleSets through a variety of techniques. The Activation should be set by calling one of the methods below and then rules can be applied as seen here.

Activate RuleSets by Name

```
// Activate ruleset by name
ruleSession.ActivateRuleSets("Entity1.RuleSet1");
```

Deactivate RuleSets by Name

```
// Deactivate ruleset by name
ruleSession.DeactivateRuleSets("Entity1.RuleSet1");
```

Activate RuleSets by Category

```
// Activate ruleset by category
ruleSession.ActivateRuleSetsByCategory("Category1");
```

Deactivate RuleSets by Category

```
// Deactivate ruleset by category
ruleSession.DeactivateRuleSetsByCategory("Category1");
```

Activate RuleSet by Name from an Entity

```
// Activate ruleset by name
entity.ActivateRuleSet("RuleSet1");
```

Deactivate RuleSet by Name from an Entity

```
// Deactivate ruleset by name
entity.DeactivateRuleSet("RuleSet1");
```

Reset RuleSet activations

```
// Reset rule activations
ruleSession.ResetAllRuleSetActivations();
```

Note: When calling ResetAllRuleSetActivations, the Activation status is set back to the original status at the time of authoring.

6.1.5.3 Executing Decisions

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime
Classes: RuleSession, RuleException
See Also: Retrieving a Rule Application, Creating a RuleSession, Handling Exceptions
References: InRule.Runtime.dll, InRule.Common.dll

Execute decision using DecisionInput

```
try
```

```
  {
        // Execute the Decision
        var decision = session.CreateDecision("CalculateArea");
       decision.Execute(new DecisionInput("height", 30), new DecisionInput("width",
400));
  }
  catch (RuleException ex)
  {
        // Handle any rule engine exceptions here.
        Console.WriteLine(ex.ToString());
  }
```

Note: If a Decision input is an Entity type, an Entity instance should be passed. Any Auto RuleSets, calculations, constraints and classifications on the passed Entity will be executed.

Execute decision using JSON

```
try
{
      // Execute the Decision
      var decision = session.CreateDecision("CalculateArea");
     decision.Execute("{ \"height\": 30, \"width\": 400 }", EntityStateType.Json));
}
catch (RuleException ex)
{
      // Handle any rule engine exceptions here.
      Console.WriteLine(ex.ToString());
}
```

6.1.5.4 Executing an Independent Rule Set or Rule Flow

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime
Classes: RuleSession, Entity
See Also: Retrieving a Rule Application, Creating a RuleSession, Creating Entities

Execute an Independent Ruleset

```
int height = 30;
int width = 40;

// Create a runtime instance of the independent ruleset.
RuleSet calcAreaRuleSet = ruleSession.CreateIndependentRuleSet("CalculateArea");

// Execute the ruleset passing in the parameters
Runtime.RuleExecutionLog executionLog = calcAreaRuleSet.Execute(height, width);
```

6.1.5.5 Checking for Notifications and Validations

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime
Classes: Notification, Validation, RuleSession
See Also: Retrieving a Rule Application, Creating a RuleSession, Creating Entities, Applying Rules

Check for notifications

```
// Retrieve the notifications from the session state
foreach (Notification note in ruleSession.GetNotifications())
{
    // Handle the notification
    Console.WriteLine(note.Message);
}
```

Check for validations

```
// Retrieve the validations from the session state
foreach (Validation validation in ruleSession.GetValidations())
{
    // Handle the validation
    Console.WriteLine(validation.Message);
}
```

6.1.5.6 Handling Exceptions

Prerequisites: A valid RuleSession, a Try block wrapped around InRule SDK code to load state and execute rules.
Namespaces: InRule.Runtime, InRule.Common.Exceptions, InRule.Repository
Classes: RuleException, CompileException, CompilerError, RuntimeException
See Also: Basic Example of Creating a RuleApplication in Code
References: InRule.Runtime.dll, InRule.Common.dll

Below are the common exceptions to handle compile errors, runtime errors and also the RuleException, which is the base class for all InRule exceptions. They are listed in the order in which they should be implemented.

The individual values in the AuthoringErrorCode and RuntimeErrorCode enumerations may change in future versions of InRule.

Note: See RuleException class for a complete list of InRule exceptions.

Handling compiler exceptions

```
// Catching compile exceptions
catch (CompileException ex)
{
    foreach (CompileError err in ex.Errors)
    {
        if (err.AuthoringErrorCode ==
AuthoringErrorCode.SqlQueryParameterTypeIsInvalid)
        {
            // React
        }
    }
}
```

Handling integration exceptions

```
// Catching integration exceptions
catch (IntegrationException ex)
{
    if (ex.RuntimeErrorCode == RuntimeErrorCode.StateUnableToBindToMember)
```

```
        {
            // React
        }
    }
```

Handling base exceptions

```
    // Base class exception; can be used to catch all InRule exceptions
    catch (RuntimeException ex)
    {
        foreach (ErrorLogMessage err in ex.ErrorMessages)
        {
            if (err.RuntimeErrorCode ==
    RuntimeErrorCode.AppSettingsSectionMissingOrMalformed)
            {
                // React
            }
        }
    }
```

6.1.5.7 Runtime Settings

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime
Classes: RuleSessionSettings, RuleSession
References: InRule.Runtime.dll

These are some of the settings the rule engine uses during execution.

Overriding the current date

Gets or sets a value that will override return value of the Today() function in the rule engine. If not set, the rule engine will use the current date.

To override the current date to January 1, 2008:

```
ruleSession.Settings.Now = new DateTime(2008, 1, 1);
```

Overriding the execution timeout

Gets or sets the maximum time for rule engine execution.

To override the default setting to 60 seconds:

```
ruleSession.Settings.ExecutionTimeout = new TimeSpan(0, 0, 60);
```

Overriding the maximum cycle count

Gets or sets the maximum cycles for rule engine execution.

To override the default setting:

```
ruleSession.Settings.MaxCycleCount = 200000;
```

Returning detailed statistics information

Gets or sets a value determining whether statistics info will be logged and returned. The default is false.

Note: This will affect the amount of information that is displayed on the Performance Statistics report.

To override the default setting:

```
ruleSession.Settings.LogOptions = EngineLogOptions.SummaryStatistics;
```

To enable metrics logging

Set the MetricLogger property to an instance of an object that implements the IMetricLogger interface as shown below.

```
ruleSession.Settings.MetricLogger = new CsvMetricLogger();
```

For more details, refer to the CSV sample application.

6.1.5.8 Retrieving and Processing the RuleExecutionLog

Prerequisites: A valid RuleSession and Entity.
Namespaces: InRule.Runtime
Classes: RuleSession, Entity
References: InRule.Runtime.dll, InRule.Common.dll

Retrieving RuleExecutionLog from RuleSession.ApplyRules

```
// Apply rules and capture RuleExecutionLog
RuleExecutionLog executionLog = ruleSession.ApplyRules();
```

Retrieving RuleExecutionLog from Entity.ExecuteRuleSet

```
// Execute explicit RuleSet and capture RuleExecutionLog
RuleExecutionLog executionLog = entity.ExecuteRuleSet("CalculatePaymentSchedule");
```

Processing the RuleExecutionLog

The following method will take a RuleExecutionLog as a parameter and return a string with the text from all of the messages separated by line breaks.

```
public string GetExecutionLogText(RuleExecutionLog executionLog)
{
    // Spin through all of the messages in the RuleExecutionLog and append to a
StringBuilder
    StringBuilder sb = new StringBuilder();
    foreach (LogMessage message in executionLog.AllMessages)
    {
        sb.Append(message.Description + Environment.NewLine);
    }
    // return string
    return sb.ToString();
}
```

See the Implementation Guide for more details.

6.1.5.9 Retrieving the Performance Log

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime
Classes: FileSystemRuleApplicationReference, RuleSession, Entity

See Also: Basic Example Of Calling Rules Engine, Retrieve the Performance Statistics Report
References: InRule.Runtime.dll, InRule.Common.dll

The Performance Log contains information about load, compile and execution times. This information is also available in the InRule Event Logs. See Event Log Details for more information. The following example demonstrates how to capture the Performance Log information using the SDK.

```
// Create the "Rectangle" entity, passing in Xml for state
Entity rectangle = session.CreateEntity("Rectangle", inputXml);

// Apply rules
session.ApplyRules();

// Get performance log information from the RuleSession
string perfLogDetails = session.Statistics.GetRunningTotalAllReport();
```

Example of Performance Log

```
SessionId: b8ea59d5-8ec3-4c8b-947c-bb41411d8304
GetRuleApplicationDefExecTime (usually indicates a compile): 14569.036ms (max
14569.036ms) 1
CreateRuleApplicationDefInfoExecTime: 1511.761ms (max 1511.761ms) 1
GetRuleApplicationRevisionKeyExecTime (incl. in CreateRuleApplicationDefInfo):
6.591ms 1
AggExecStatInfo.AggDirectives (incl. in Submit): 399.384ms 1
CreateSessionExecTime: 93.793ms 1
CreateEntityExecTime: 17511.331ms (max 17511.331ms) 1
LoadXmlExecTime: 93.128ms 1
SubmitExecTime: 642.903ms (max 642.903ms) 1
ProcessResponseExecTime: 22.564ms 1
ThreadId: 11
MaxCacheDepth: 5
CurrentCacheDepth: 1
CompileExecTime: 17195.075ms 1
WorkingMemCreateCnt: 1
Ruleapp 'RectangleApp': 1
CacheUpTime: 18.421sec
RunningTotalAll: 35558.794ms
```

6.1.5.10 Retrieving the Performance Statistics Report

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime
Classes: FileSystemRuleApplicationReference, RuleSession, Entity, RuleException
See Also: Basic Example Of Calling Rules Engine, Retrieve the Performance Log
References: InRule.Runtime.dll, InRule.Common.dll

The following basic example demonstrates how to capture the Performance Statistics Report at run time.

```
// Turn on capturing of detailed statistics (if desired)
ruleSession.Settings.LogOptions = EngineLogOptions.DetailStatistics;

// Run rules
ruleSession.ApplyRules();

// Get the XML as a string for the Performance Statistics Report (includes log
messages if log options configured to include them)
string reportXml = ruleSession.LastRuleExecutionLog.GetXml();
```

```
// Get the HTML as a string for the Performance Statistics Report
string reportHtml = ruleSession.LastRuleExecutionLog.GetHtml();

// write the report to the file system (could also view in a web browser control)
File.WriteAllText(@"c:\work\PerfStatsReport.html", reportHtml);
```

6.1.6 Working with the Rule Application Cache

Working with the Rule Application Cache

- Adding Items into the Cache
- Managing the Cache
- Controlling Compilation and Cache Retention
- Iterating Items in the Cache

6.1.6.1 Adding Items into the Cache

Prerequisites:
Namespaces: InRule.Runtime
Classes: FileSystemRuleApplicationReference, CatalogRuleApplicationReference, InMemoryRuleApplicationReference, RuleSession
See Also: Creating a RuleSession with Cache Retention, Controlling Compilation and Cache Retention, Iterating items in the cache , Working with the Rule Application Cache
References: InRule.Runtime.dll, InRule.Common.dll

In general, items are automatically added to the cache when needed or explicitly using Compile, however you can manually add rule applications into the cache:

```
string ruleAppPath = @"c:\temp\Invoice.ruleApp";

// add an item into the cache using a rule application reference
var ruleApp = new FileSystemRuleApplicationReference(ruleAppPath);
RuleSession.RuleApplicationCache.Add(ruleApp);

// add an item into the cache using the path, notice that the same options provided
// by the compile method are also available
RuleSession.RuleApplicationCache.Add(ruleAppPath, true,
CacheRetention.FromWeight(3000));
```

6.1.6.2 Managing the Cache

Prerequisites: A valid RuleApplicationReference
Namespaces: InRule.Runtime
Classes: RuleSession, RuleApplicationReference, FileSystemRuleApplicationReference, InMemoryRuleApplicationReference, CatalogRuleApplicationReference
See Also: Retrieving a Rule Application, Creating a RuleSession, Creating a RuleSession with Cache Retention, Adding items into the Cache, Iterating items in the cache , Controlling Compilation and Cache Retention

Compiling a rule application with delegates

When you compile your rule application, you can specify if executable code (delegates) will be compiled at the same time or only when needed. If you select the former, there will be a slower compile time, but the execution time will be faster.

```
// Compile the RuleApplicationReference, including the executable code (delegates)
ruleAppRef.Compile(CompileSettings.Default);
```

Setting the degree of parallelism for compilation

You can direct InRule regarding how many threads or cores to use for parallel work when compiling a rule application. This value is a non-negative integer, which you set as follows:

Value	Meaning
0	on; InRule will decide based on the number of cores available
1	off; one thread will be used for all work (default)
n	the number of threads to use

```
// Compile the RuleApplicationReference, specifying at most two threads or cores
ruleAppRef.Compile(CompileSettings.Create(EngineLogOptions.None, 2));
```

Note: InRule recommends the default setting (0).

Adding a rule application to the cache

After compiling your rule application, you can speed up future executions by adding it to the rule session cache.

```
// Add the rule application to the rule application cache
RuleSession.RuleApplicationCache.Add(ruleAppRef);
```

Removing a rule application from the cache

Of course, you may also remove a rule application from the cache.

```
// Remove a rule application from the rule application cache
RuleSession.RuleApplicationCache.Remove(ruleAppRef);
```

Clearing the rule application caches

From time to time, you may want to clear the caches completely. You may do this for the internal cache of a rule application, or for the entire rule session cache.

```
// Clear internal caches in a rule application reference
ruleAppRef.ClearCompiledFunctions();

// Clear the entire rule application cache
RuleSession.RuleApplicationCache.Clear();
```

6.1.6.3 Controlling Compilation and Cache Retention

Prerequisites:
Namespaces: InRule.Runtime
Classes: RuleApplicationReference
See Also: Creating a RuleSession with Cache Retention, Adding items into the Cache, Iterating items in the cache , Working with the Rule Application Cache
References: InRule.Runtime.dll, InRule.Common.dll

```
// You can check to see if a rule application is compiled
if (!ruleApp.IsCompiled)
{
    // Compile functions as well as metadata, if false only the
    // metadata is compiled
    bool compileFunctions = true;

    // Set the cache retention, AlwaysRetain will extend the cache to fit all
    // apps compiled with AlwaysRetain
    CacheRetention cacheRetention = CacheRetention.AlwaysRetain;

    // You can also use a weight to control how an application is retained in the
    // cache, a high weight take preference over lower weights - int.MaxValue is
the
    // same as Always Retain
    cacheRetention = CacheRetention.FromWeight(2000);

    // Compile the rule app
    ruleApp.Compile(compileFunctions, cacheRetention);

}
```

6.1.6.4 Iterating Items in the Cache

Prerequisites:
Namespaces: InRule.Runtime
Classes: RuleApplicationReference
See Also: Creating a RuleSession with Cache Retention, Controlling Compilation and Cache Retention, Adding Items into the Cache, Working with the Rule Application Cache
References: InRule.Runtime.dll, InRule.Common.dll

You can iterate through all the cache entries to get information:

```
public string GetCacheEntriesInfo()
{
    var sb = new StringBuilder();
    // Get the existing cache entries from the RuleSession - this returns
    // a set of RuleApplicationReferences
    IEnumerable<RuleApplicationReference> cacheEntries;
    cacheEntries = RuleSession.RuleApplicationCache.Items;
    foreach (RuleApplicationReference cacheEntry in cacheEntries)
    {
        // Extract information from the rule application
        sb.AppendLine("Rule App Name:" + cacheEntry.GetRuleApplicationDef().Name);
        sb.AppendLine("Unique Name:" + cacheEntry.Name);

        // if it has been compiled, you can get properties such as time
        if (cacheEntry.LastMetadataCompile != null)
        {
            DateTime lastMetaCompileTime =
cacheEntry.LastMetadataCompile.Value.UtcDateTime;
               sb.AppendLine("Last metadata compile:" +
lastMetaCompileTime.ToString());
          }
          // get information about its rank in the cache
          sb.AppendLine("Cache retention rank:" +
cacheEntry.CacheRetention.Weight.ToString());
          sb.AppendLine();
     }
     return sb.ToString();
}
```

6.1.7 Working with Rule Application Metadata

Working with Rule Application Metadata

- Using the RuleApplicationDef Object
- Retrieving Definition Objects at Runtime
- Using Element Metadata
- Working with Attributes
- Working with Value Lists

6.1.7.1 Using the RuleApplicationDef Object

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime
Classes: RuleApplicationDefInfo, RuleApplicationDef, DataElementDef, EntityDef
See Also: Creating a RuleSession

RuleApplicationDef metadata

```
// System and user-defined attributes
string attributeValue =
ruleSession.GetRuleApplicationDef().Attributes["MyAttribute"];

// Categories defined in the rule application
CategoryDefCollection categories = ruleSession.GetRuleApplicationDef().Categories;

// Getting name of first category in collection
string catalogName = categories[0].Name;

// Getting DataElementDef
DataElementDef dataElementDef = (DataElementDef)
ruleSession.GetRuleApplicationDef().DataElements["MyInlineTable"];
```

Iterate through the entities of a rule application

```
// Iterate through the entities for the rule application
```

```
foreach (EntityDef entityDef in ruleSession.GetRuleApplicationDef().Entities)
{
    Console.WriteLine(entityDef.Name);
}
```

6.1.7.2 Retrieving Definition Objects at Runtime

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime, InRule.Repository, InRule.Repository.RuleElements
Classes: EntityDef, FieldDef, RuleSetDef, RuleElementDef, Entity, Field, RuleSet, RuleElement
See Also: Using Element Metadata

The following examples demonstrate how to retrieve definition objects through the runtime objects. These objects can then be utilized to access object metadata and user-defined attributes.

```
// Get EntityDef from Entity
EntityDef entityDef = entity.GetDef();

// Get FieldDef from Field
FieldDef fieldDef = field.GetDef();

// Get CollectionDef from Collection, assigned to FieldDef
FieldDef collectionDef = collection.GetDef();

// Get RuleSetDef from RuleSet
RuleSetDef ruleSetDef = (RuleSetDef)ruleSet.GetDef();

// Get RuleElementDef from RuleElement, getting first child element in this example
RuleElementDef ruleElementDef = ruleSet.RuleElements[0].GetDef();
```

6.1.7.3 Using Element Metadata

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime, InRule.Repository, InRule.Repository.RuleElements
Classes: EntityDef, FieldDef, RuleSetDef, RuleElement
See Also: Retrieving Definition Objects at Runtime

Element definition and user-defined metadata can be retrieved from the definition objects, which are available at runtime. See Retrieving Definition Objects at Runtime for details.

Iterate through the fields objects on an entity

```
// Iterate through fields in an entity
foreach (FieldDef fieldDef in invoiceEntity.GetDef().Fields)
{
    if (fieldDef.IsCalculated)
        // Display the expression if a calculation
        Console.WriteLine(fieldDef.Calc.FormulaText);

    if (fieldDef.IsAnEntityDataType)
        // Display the referenced entity if a child entity field
        Console.WriteLine(fieldDef.DataTypeEntityName);
}
```

Field metadata

```
// Get data type
string dataType = fieldDef.DataType.ToString();

// Get default value
string defaultValue = fieldDef.DefaultValue.ToString();

// Get Attributes
ICollection attributes = fieldDef.Attributes.Values;

// Get assigned categories
ICollection assigned = fieldDef.AssignedCategories;
```

Iterate through the rulesets on an entity

```
// Iterate through the rule sets on an entity
foreach (RuleSet ruleSet in invoiceEntity.RuleSets)
{
    // Cast into a RuleSetDef
    RuleSetDef ruleSetDef = (RuleSetDef)ruleSet.GetDef();
    // See if this is an active explicit ruleset
    if (ruleSetDef.FireMode == RuleSetFireMode.Explicit && ruleSetDef.IsActive)
    {
        Console.WriteLine(ruleSet.Name);
    }
}
```

6.1.7.4 Working with Attributes

Prerequisites: A valid EntityDef and FieldDef
Namespaces: InRule.Repository
Classes: EntityDef, FieldDef
See Also: Creating a RuleSession, Retrieving Definition Objects at Runtime

Retrieve a user-defined attribute on an entity

```
// Retrieve an attribute defined on an entity
Console.WriteLine(entityDef.Attributes.Default["MyAttribute"]);
```

Iterate through the attributes on a field

```
if (fieldDef.Attributes != null && fieldDef.Attributes.Count > 0)
{
    foreach (XmlSerializableStringDictionary.XmlSerializableStringDictionaryItem
attributeItem in fieldDef.Attributes.Default)
    {
        Console.WriteLine(attributeItem.Key + ":" + attributeItem.Value);
    }
}
```

6.1.7.5 Working with Value Lists

Prerequisites: A valid Entity and RuleSession

Namespacesxxx: InRule.Runtime, InRule.Repository
Classes: ValueListItem, RuleApplicationDef, ListItemDefCollection

Populating a combobox from a value list

```
// Retrieve a value list that is associated to a field
ValueList valueList = entity.Fields["State"].AssociatedValueList;

foreach (ValueListItem valueListItem in valueList)
{
    // Add the list item value to the combobox
    comboBox.Items.Add(valueListItem);
    comboBox.DisplayMemberPath = "Value";
}
```

Populating a combobox from a standalone value list

```
// Get value list using the session's DataManager
ValueList myList = ruleSession.Data.GetValueList("StandAloneValueList");

// Tell combo box what field to display to the end user
comboBox.DisplayMemberPath = "DisplayText";

// Add the item to the combo box
foreach (ValueListItem item in myList)
{
    comboBox.Items.Add(item);
}
```

6.1.8 Other

Other

- Launching irVerify or the StateViewer From Code
- Overriding EndPoints at Runtime
- Overriding DataElements at Runtime
- Overriding Culture Settings at Runtime
- Rule Tracing Input and Output Using RuleSession
- Working with the Trace Viewer Through irSDK
- Executing a Simple Test Suite

6.1.8.1 Launching irVerify and the State Viewer from code

Prerequisites: A valid RuleSession and Entity.
Namespaces: InRule.Runtime.Testing
Classes: EntityTester, EntityStateViewer
See Also: Retrieving a Rule Application, Creating a RuleSession, Creating Entities

Start irVerify

```
// Start irVerify by passing in the entity
EntityTester.Show(mortgageEntity);
```

Show the state viewer

```
// Show the state viewer by passing in the entity
EntityStateViewer.Show(mortgageEntity);
```

Required System References

- PresentationFramework
- PresentationCore
- WindowsBase
- System.Xaml

Required InRule References

- inrule.runtime.dll
- inrule.runtime.testing.dll
- inrule.common.dll

6.1.8.2 Overriding EndPoints at Runtime

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime, InRule.Repository.EndPoints
Classes: RuleSession, DatabaseConnection, SendMailServerDef, WebServiceDef
See Also: Retrieving a Rule Application, Creating a RuleSession

Setting the database endpoint

```
// Override InvoiceDB connection string
string connString =
ConfigurationManager.ConnectionStrings["TestDb"].ConnectionString;
ruleSession.Overrides.OverrideDatabaseConnection("InvoiceDB", connString);
```

Setting the WebService endpoint

```
// Override the web service address
ruleSession.Overrides.OverrideWebServiceAddress("WebService1", "http://localhost/
MyWebService/service1.svc?wsdl");

// Override the web service MaxReceivedMessageSize
ruleSession.Overrides.OverrideWebServiceMaxReceivedMessageSize("WebService1",
Int32.MaxValue);
```

Setting the Mail server endpoint

```
//Override the smtp server from the .config file setting
string server = ConfigurationManager.AppSettings["MailServer"];
ruleSession.Overrides.OverrideMailServerConnection("SmtpServer", server);
```

6.1.8.3 Overriding DataElements at Runtime

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime, InRule.Repository
Classes: RuleSession, TableDef, XmlDocumentDef
See Also: Retrieving a Rule Application, Creating a RuleSession

Overriding an inline table

```
ValueList newValueList = ValueList.New(
    "Mon", "Monday",
    "Tue", "Tuesday",
    "Wed", "Wednesday",
    "Thu", "Thursday",
    "Fri", "Friday",
    "Sat", "Saturday",
    "Sun", "Sunday");
ruleSession.Overrides.OverrideInlineValueList("InlineValueList1", newValueList);
```

Overriding an inline XML document

```
using (var reader = new StringReader("<invoice><lineitem><id>1</id><name>Pencil</
name><price>$1.00</price></lineitem><lineitem><id>2</id><name>Computer</
name><price>$1000.00</price></lineitem></invoice>"))
{
    XPathDocument xml = new XPathDocument(reader);

    // Override the inline XML document
    ruleSession.Overrides.OverrideXmlDocument("InlineXMLDocName", xml);
}
```

Overriding an SQL query

```
ruleSession.Overrides.OverrideQuery("SQLQuery1", "select Name from Customer where
id = 2");
```

6.1.8.4 Overriding Culture Settings at Runtime

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime
Classes: RuleSession
See Also: InRule Culture Settings

The following using statements are required

```
using System.Threading;
using System.Globalization;

// Create CultureInfo object, passing in desired culture
CultureInfo newCulture = new CultureInfo("fr-FR");

// Assign to current thread before applying rules
Thread.CurrentThread.CurrentCulture = newCulture;

// Apply rules
```

```
ruleSession.ApplyRules();
```

6.1.8.5 Rule Tracing Input and Output through irSDK

Prerequisites: none
Namespaces: InRule.Runtime, InRule.Runtime.Tracing.Events, InRule.Runtime.Tracing.Frames
Classes: FileSystemRuleApplicationReference, RuleSession, Entity
See Also: Working with the Trace Viewer through irSDK, Basic Example of Calling the Rule Engine, Creating a RuleSession, Creating Entities

The following code sample shows methods to extract rule tracing output to various *.ruletrace and *.xml formats. The *.ruletrace files can be opened directly in irAuthor so that trace data can be evaluated in the trace viewer.

Tracing is enabled by setting the LogOptions Property to EngineLogOptions.RuleTrace.

During execution, a temporary file is used to persist tracing data. This file is cleaned up by the Dispose method on the RuleSession. However, copies of the file can be persisted using the WriteExportTrace method as shown in the example below.

```
RuleApplicationReference ruleAppRef = new FileSystemRuleApplicationReference(@"C:
\RuleApps\MortgageCalculator.ruleapp");

using (var session = new RuleSession(ruleAppRef))
{
    session.Settings.LogOptions = EngineLogOptions.RuleTrace;

    var mortgageEntity = session.CreateEntity("Mortgage");
    mortgageEntity.LoadXml(@"C:\RuleApps\Mortgage.xml");
    RuleExecutionLog log = null;
    try
    {
            // Temp files are created for session during apply/execute rules,
cleaned on dispose of session
            log = session.ApplyRules();
    }
    catch (RuntimeException ex)
    {
            // In the event of an error, the execution trace can still be retrieved
from the RuntimeException
            log = ex.Log;
    }

    // Copies of trace files are generated when trace is created during
GetExecutionTrace, and cleaned on dispose of trace.
    // Make sure the IExecutionTrace is disposed, so trace files are cleaned up
    using (var trace = log.GetExecutionTrace())
    {
            // Writes out a complete "rule trace" package that includes the rule
application and trace data.
            // This *.ruletrace file can be opened in irAuthor
            trace.WriteExportPackage(@"C:\temp\mylog.ruletrace");

            // Write out the trace frame XML
            trace.WriteXml(@"C:\temp\mylog.frames.xml", new
TraceFrameXmlWriterSettings());
```

```
          // Write out the events XML
          trace.EventReader.GetAllEvents().WriteXml(@"C:\temp\mylog.events.xml",
new TraceEventXmlWriterSettings());

          // Write out events in a tab-delimited format that can be viewed in
Excel
          trace.EventReader.GetAllEvents().WriteTabDelimited(@"C:\temp
\mylog.events.txt"); // same as clipboard

          // Get only specific types of events
          // Processed as a "SQL LIKE statement"
          trace.EventReader.GetFilteredEvents("filtertext").WriteXml(@"C:\temp
\myfilteredlog.events.xml", new TraceEventXmlWriterSettings());

          // Retrieve a "page" of event data
          var firstPage = trace.EventReader.GetAllEvents().GetEventPage(1, 100);
    }
}

// Rule execution trace can be populated from a previously persisted file
var executionTrace = ExecutionTrace.Load(@"C:\temp\mylog.ruletrace");
```

6.1.8.6 Working with the Trace Viewer through irSDK

Prerequisites: none
Namespaces: InRule.Runtime, InRule.Runtime.Testing
Classes: FileSystemRuleApplicationReference, RuleSession, Entity
See Also: Rule Tracing Input and Output through irSDK, Basic Example of Calling the Rule Engine, Creating a RuleSession

The Trace Viewer used with irAuthor and irVerify is also accessible from the SDK. It is available as a popup Window and an embeddable control.

Using the Trace Viewer Window

```
using (var trace = log.GetExecutionTrace())
{
    // Call static method to launch default trace viewer
    ExecutionTraceWindow.Show(trace);

    //If any configuration of the window is required, the static method returns the
window object,
    // which can be used to set configuration options as shown here.

    // Static method returns reference window to the window in case further
configuration is required
    var window = ExecutionTraceWindow.Show(trace);

    // Hide export window
    window.ShowExportMenu = false;

    // Reusing same detail panel
    window.DetailPopupMode = TraceEventDetailPopupMode.ShowInExistingWindow;
}
```

Embedding the Trace Viewer Control

```
var log = ruleSession.LastRuleExecutionLog;
// Create an instance of the control
var view = new ExecutionTraceView();

// Specify if you want to see the Export menu (default is visible)
view.ShowExportMenu = false;

// Specify if you want the detail windows to be reused or to create a new window
each time
// Details window is displayed when clicking hyperlink or double clicking on the
row
view.DetailPopupMode = TraceEventDetailPopupMode.ShowInNewWindow;

// Load the trace into the viewer
// Options include IExecutionTrace, ExecutionLog (used here) or path to trace file
view.Load(log);
```

6.1.8.7 Executing a Simple Test Suite

Prerequisites: A valid test suite
Namespaces: InRule.Runtime, InRule.Runtime.Testing.Regression, InRule.Runtime.Testing.Session
Classes: TestSuiteDef, TestResultCollection, TestingSessionManager, RegressionTestingSession
See Also: Authoring a Simple Test Suite for Regression Testing

The key points in the below code sample, are that the TestingSessionManager should be used with a using() pattern to ensure it is disposed when testing is finished. This ensures any lingering RuleSessions are also disposed correctly.

```
public static void ExecuteSimpleTestSuite()
{
    // Get the TestSuitePersistenceProvider from the file system
    TestSuitePersistenceProvider testProvider = new
ZipFileTestSuitePersistenceProvider(@"C:\Temp\SimpleTestSuite.testsuite");

    // Load the TestSuiteDef using the provider
    TestSuiteDef suite = TestSuiteDef.LoadFrom(testProvider);

    // Load the rule application into the test suite
    suite.ActiveRuleApplicationDef = RuleApplicationDef.Load(@"C:\Temp
\SimpleRuleApp.ruleapp");

    // Set up a testing manager with an InProcessConnection factory
    using (TestingSessionManager manager = new TestingSessionManager(new
InProcessConnectionFactory()))
    {
            // Create the testing session
            RegressionTestingSession session = new RegressionTestingSession(manager,
suite);

            // Execute all Tests in the Test Suite - Ensure results collection is
disposed
```

```
          using (TestResultCollection results = session.ExecuteAllTests())
          {
                 // Persist TestResults to the file system
                 results.SaveAs(@"C:\Temp\SimpleTestSuiteResults.testresults");

                 // Output result information
                 Console.WriteLine("{0} Test(s) executed.", results.Count);
                 Console.WriteLine("TestResult 1 {0}.", results[0].Passed ?
"passed" : "failed");
                 Console.WriteLine("TestResult 1 execution duration: {0}.",
results[0].Duration);
                 Console.WriteLine("TestResult 1, Assertion 1 Expected Value:
{0}.", results[0].AssertionResults[0].FormattedExpectedValue);
                 Console.WriteLine("TestResult 1, Assertion 1 Actual Value: {0}.",
results[0].AssertionResults[0].FormattedActualValue);
                 Console.WriteLine("TestResult 1, Assertion 1 Target: {0}.",
results[0].AssertionResults[0].Target);
                 Console.WriteLine("TestResult 1, Assertion 1 Display Text: {0}.",
results[0].AssertionResults[0].DisplayText);
          }
    }
}
```

The output looks like the following:

```
1 Test(s) executed.
TestResult 1 passed.
TestResult 1 execution duration: 00:00:00.2921900.
TestResult 1, Assertion 1 Expected Value: 7.
TestResult 1, Assertion 1 Actual Value: 7.
TestResult 1, Assertion 1 Target: Field2.
TestResult 1, Assertion 1 Display Text: Field2 is equal to 7.
```

6.1.8.8 Logging Metrics

Prerequisites: A valid RuleSession
Namespaces: InRule.Runtime
Classes: IMetricLogger, RuleSession
See Also: Retrieving a Rule Application, Creating a RuleSession

To handle the multitude of options for desired logging locations, InRule has implemented an adaptor based model. An adaptor is a .NET assembly that is available to the rule engine that implements the IMetricLogger interface. When this assembly exists, the engine will call out to the required methods in the assembly to perform the actual logging. This provides customers the ability to write metrics to any location that is required in their implementation.

At the time of this writing, there are 3 adaptors that are available in GitHub.

- Microsoft® Azure® Table Storage
- SQL Server
- CSV (primarily for demo purposes)

To enable metrics logging

You must set the MetricLogger property in the Rule Session Settings to an instance of an object that implements the IMetricLogger interface.

```
ruleSession.Settings.MetricLogger = new CsvMetricLogger();
```

Example of a Metrics Logger which logs to a CSV file

```
public sealed class CsvMetricLogger : IMetricLogger
{
    public async Task LogMetricsAsync(string serviceId, string ruleApplicationName,
Guid sessionId, Metric[] metrics)
    {
        // for async sample see the Azure Table Storage Adaptor
        throw new NotImplementedException();
    }

    public void LogMetrics(string serviceId, string ruleApplicationName, Guid
sessionId, Metric[] metrics)
    {
        // loop through all of the metrics that are emitted by the rules engine
        // there will be one metric per entity
        foreach (Metric metric in metrics)
        {
            // get a list of the field and/or rule names
            var fields = new List<string>();
            foreach (var metricProperty in metric.Schema.Properties)
            {
                fields.Add(metricProperty.Name);
            }

            // get the value of each field or rule
            var values = new List<string>();
            foreach (var metricProperty in metric.Schema.Properties)
            {
                values.Add(metric[metricProperty].ToString());
            }

            // save them to disk in a csv file
            SaveToFile(fields, values, metric.EntityName);
        }
    }

    private void SaveToFile(List<string> fields, List<string> values, string
metricEntityName)
    {
        var fileName = $"{ConfigurationManager.AppSettings["OutputDirectory"]}
{metricEntityName}.csv";

        if (File.Exists(fileName))
        {
            // if the file already exists, append the values
            File.AppendAllText(fileName, string.Join(",", values.ToArray()) +
Environment.NewLine);
        }
        else
        {
            // if the file doesn't exist, create it and include the headers
            var s = new StringBuilder();
            s.AppendLine(string.Join(",", fields.ToArray()));
            s.AppendLine(string.Join(",", values.ToArray()));
            File.WriteAllText(fileName, s.ToString());
        }
        }
}
```

For more details, refer to the CSV metrics logger sample application.

6.2 Authoring API Examples

The InRule Authoring API (also referred to as the Repository SDK) spans two functional areas: developing against the RuleApplicationDef model and embedding authoring controls.

Developing against the Rule Application definition object model

- Working with a RuleApplicationDef for Authoring
- Basic Example of Creating a Rule Application in Code
- Working With RuleSets
- Dynamically Generating a RuleApplication Schema
- Modifying EndPoints
- Authoring a UDF in Code
- Inline Table and Value Lists

Working with Advanced Definition Objects

- Removing Templates in the Language Editor
- Determine FieldDef Dependencies

Working with the Regression Tester SDK

- Authoring a Simple Test Suite for Regression Testing

Embedding Authoring Controls

- Embedding InRule Default Editors
- Embedding the Language Rule Editor
- Embedding the Decision Table Editor
- Embedding the Condition Editor

6.2.1 Developing against the RuleApplicationDef Object Model

Developing against the Rule Application definition object model

- Working with a RuleApplicationDef for Authoring
- Basic Example of Creating a Rule Application in Code
- Working With RuleSets
- Dynamically Generating a RuleApplication Schema
- Modifying EndPoints
- Authoring a UDF in Code
- Inline Table and Value Lists

6.2.1.1 Working with a RuleApplicationDef for Authoring

Prerequisites: None
Namespaces: InRule.Repository
Classes: RuleApplicationDef, RuleAppRef, RuleCatalogConnection
See Also: Opening a RuleApplicationDef for Authoring from Catalog, Working with RuleApplicationDef in the Catalog

The RuleApplicationDef object is used to dynamically create and modify rules and schema elements in code.

Create new rule application

```
RuleApplicationDef ruleAppDef = new RuleApplicationDef();
```

Load existing rule application from file system

```
RuleApplicationDef ruleAppDef = RuleApplicationDef.Load(@"C:\RuleApps
\MortgageCalculator.ruleapp");
```

Save a rule application to the file system

```
ruleAppDef.SaveToFile(@"C:\RuleApps\MortgageCalculator.ruleapp");
```

6.2.1.2 Basic Example of Creating a Rule Application in Code

Prerequisites: None
Namespaces: InRule.Repository, InRule.Repository.RuleElements
Classes: RuleApplicationDef, EntityDef, FieldDef, RuleSetDef, SimpleRuleDef, FireNotificationActionDef
References: InRule.Repository.dll

The RuleApplicationDef object is used to dynamically create and modify rules and schema elements in code. The example below creates the simple rectangle rule application.

```
// Create a new rule application named "RectangleApp"
RuleApplicationDef ruleAppDef = new RuleApplicationDef("GeneratedRectangleApp");

// Create a rectangle entity
EntityDef entityDef = new EntityDef("Rectangle");

// Add the rectangle entity the rule application's entity collection
ruleAppDef.Entities.Add(entityDef);

// Add numeric height field
entityDef.Fields.Add(new FieldDef("Height", DataType.Number));

// Set the height default value to 4
entityDef.Fields["Height"].DefaultValue = "4";

// Add numeric width field
entityDef.Fields.Add(new FieldDef("Width", DataType.Number));

// Set the width default value to 5
```

```
entityDef.Fields["Width"].DefaultValue = "5";

// Add numeric area calculation with syntax expression
entityDef.Fields.Add(new FieldDef("Area", "Height * Width", DataType.Number));

// Create a ruleset
RuleSetDef ruleSetDef = (RuleSetDef)entityDef.RuleElements.Add(new
RuleSetDef("MeasurementRules"));

// Create a simple if then rule with condition expression
SimpleRuleDef ruleDef = new SimpleRuleDef("Height > 0 and Width > 0");
ruleSetDef.Rules.Add(ruleDef);

// Create a notification with tokenized message
FireNotificationActionDef notificationDef = new FireNotificationActionDef();
notificationDef.NotificationMessageText = "Generated RectangleApp: Height <%Height
%> * Width <%Width%>  = Area <%Area%>";

// Add the notification as an action under the if then rule
ruleDef.SubRules.Add(notificationDef);

// Save the rule application to the file system
ruleAppDef.SaveToFile(@"c:\temp\Rectangle.ruleapp");
```

6.2.1.3 Working with RuleSets

Prerequisites: A valid RuleApplicationDef and EntityDef
Namespaces: InRule.Repository
Classes: RuleApplicationDef, EntityDef, RuleSetDef
References: InRule.Repository.dll
See Also: Dynamically Generating a Rule Application Schema

The following code samples demonstrate how to retrieve and work with RuleSets for authoring purposes.

Get a list of RuleSets from an EntityDef

```
// This will return all top level RuleSets off of an Entity
RuleSetDef[] ruleSets = entityDef.GetRuleSets();
```

Get a list of all RuleSets from an EntityDef

```
  // This will return all RuleSets off of an Entity, including RuleSets that are
under Rule Folders
  RuleSetDef[] ruleSets = entityDef.GetAllRuleSets();
```

Get a specific RuleSet by name from an EntityDef

```
// This will return a single RuleSetDef
RuleSetDef ruleSetDef = entityDef.GetRuleSet("MyRuleSet");
```

Get a list of independent RuleSets

```
// This will return a collection of independent RuleSets
RuleSetDefBaseCollection indRuleSets = ruleAppDef.RuleSets;
```

Get a specific independent RuleSet

```
// This will return a collection of independent RuleSets
RuleSetDef indRuleSet = ruleAppDef.GetRuleSet("MyIndRuleSet");
```

Enable/Disable a RuleSet

```
// This will turn off the RuleSet, making it unavailable at runtime
// A disabled RuleSet cannot be turned on at runtime
ruleSetDef.IsActive = false;
```

Set Default Activation of a RuleSet

```
  // This will turn off the default activation for the RuleSet
  // A RuleSet that is enabled, can be activated or deactivated at runtime via rules
or the SDK
  ruleSetDef.DefaultActivation = false;
```

Set RuleSet Run Mode

```
// This will set the run mode to sequential
ruleSetDef.RunMode = RuleSetRunMode.Sequential;
```

Set RuleSet Fire Mode

```
// This will set the fire mode to explicit
ruleSetDef.FireMode = RuleSetFireMode.Explicit;
```

6.2.1.4 Dynamically Generating a Rule Application Schema

Prerequisites: A valid RuleApplicationDef
Namespaces: InRule.Repository, InRule.Repository.EndPoints, InRule.Repository.ViewsAndControllers
Classes: RuleApplicationDef, XmlSchemaDef, XmlSchemaDefController, AssemblyDef, AssemblyDefController
References: InRule.Repository.dll

The following examples demonstrate how to dynamically generate a rule application schema using a .NET assembly and an XSD.

Generate Rule Application Schema using an XSD

```
// Create the schema def object using a name for the schema and the path to the Xsd
file
XmlSchemaDef schemaDef = new XmlSchemaDef("XsdSchemaName", FilePath +
"invoice.xsd");

// Embed the schema in the rule app (optional)
schemaDef.UseEmbeddedXsd = true;

// Set whether validation should run when loaded (default is true)
schemaDef.EnableXsdValidation = true;

// Add the schema def object to the rule application as an end point
ruleAppDef.EndPoints.Add(schemaDef);

// Create the schema controller which will do the import of the Xsd
XmlSchemaDefController controller = new XmlSchemaDefController(schemaDef);
```

```
// Do the import
controller.Import(FilePath + "invoice.xsd");

// Generate the rule application schema
string[] applyWarns = controller.Apply();

// Process warnings if there were any
if (applyWarns.Length > 0)
{
    // handle warnings/errors
}

// Bind Fields that map to xs:enumeration restriction types to a ValueList during
import
schemaDef.BindEnumerationFieldsToValueLists = true;

// Disable creation of Constraint rules for Fields that map to xs:enumeration
restriction types during import
schemaDef.CreateConstraintsForEnumerationFields = false;
```

Generate Rule Application Schema using a .NET Assembly

```
// Create Assembly and Assembly controller def objects, set isSchema to true
AssemblyDef assemblyDef = new AssemblyDef("InvoiceObject", true);
AssemblyDefController assemblyDefController = new
AssemblyDefController(assemblyDef);

// Import the assembly, notes can be captured during import
string[] importNotes =
assemblyDefController.Import("InvoiceObjects.dll").EntityDefsInfo.Notes;

// Get top level class by alias
AssemblyDef.ClassInfo topLevelClass =
assemblyDef.ClassInfos.GetByAliasName("Invoice");

// Select all dependent entities for the top level class
assemblyDefController.CheckAllSelectedDependentEntities(topLevelClass);

// Add the assembly to the EndPoints collection (will appear as schema in irAuthor)
ruleAppDef.EndPoints.Add(assemblyDef);

// Generate the schema, notes can be captured during schema generation
string[] applyNotes = assemblyDefController.Apply();

// Bind Fields that map to Enum types to a ValueList during import
assemblyDef.BindEnumerationFieldsToValueLists = true;

// Disable creation of Constraint rules for Fields that map to Enum types during
import
assemblyDef.CreateConstraintsForEnumerationFields = false;
```

Generate Rule Application Schema using a Database

```
// Create new end point
DatabaseConnection sch = (DatabaseConnection)ruleAppDef.EndPoints.Add(new
DatabaseConnection("DbConn1", connectionString));

// Set up end point to be a schema
```

```
sch.IsSchemaDefining = true;

// Create controller that will perform the import
DatabaseConnectionController controller = new DatabaseConnectionController(sch);

// Do the import
controller.Import();

// Apply changes to the rule app, catching any errors that may have occurred
string[] errors = controller.Apply();
```

6.2.1.5 Modifying EndPoints

Prerequisites: A valid RuleApplicationDef
Namespaces: InRule.Repository, InRule.Repository.EndPoints
Classes: RuleApplicationDef, EndPointDef, DatabaseConnection
References: InRule.Repository.dll
See Also: Dynamically Gererating a Rule Application Schema

The following code sample demonstrates how to modify the connection string of a Database Connection EndPoint.

```
// Get endpoint from loaded RuleApplicationDef object
EndPointDef endPointDef = ruleAppDef.EndPoints["DatabaseConnection1"];

// Cast into a DatabaseConnection object
DatabaseConnection dbConn = (DatabaseConnection)endPointDef;

// Update the connection string
dbConn.ConnectionString = newConnectionString;
```

6.2.1.6 Authoring a UDF in Code

Prerequisites: None
Namespaces: InRule.Repository, InRule.Repository.RuleElements
Classes: RuleApplicationDef, EntityDef, FieldDef, RuleSetDef, UdfLibraryDef, UdfDef, UdfArgumentDef, CalcDef, ExecuteMethodActionDef, ExecuteMethodActionParamDef
References: InRule.Repository.dll

The sample code snippet below illustrates authoring an UDF and using an ExecuteMethodActionDef to execute the UDF.

```
// Create a ruleapplication
RuleApplicationDef ruleAppDef = new RuleApplicationDef();
EntityDef entity1 = ruleAppDef.Entities.Add(new EntityDef("Entity1"));
entity1.Fields.Add(new FieldDef("InputText"));
entity1.Fields.Add(new FieldDef("Pattern"));

// Add a UDF library
UdfLibraryDef udfLibrary = new UdfLibraryDef("StringFunctionsLibrary");

// Define UDF to match a string for a given pattern
UdfDef udf = new UdfDef("IsPatternMatch");

// Input arguments
UdfArgumentDef arg1 = new UdfArgumentDef("InputText");
```

```
arg1.ArgumentTypeInfo.DataType = DataType.String;
udf.FunctionArguments.Add(arg1);
UdfArgumentDef arg2 = new UdfArgumentDef("Pattern");
arg2.ArgumentTypeInfo.DataType = DataType.String;
udf.FunctionArguments.Add(arg2);

// Return type
udf.ReturnTypeInfo.DataType = DataType.Boolean;

// Function declaration
CalcDef func = new CalcDef();
func.FormulaText = "return InputText.Contains(Pattern);";
udf.FunctionBody = func;

// Add UDF to the library
udfLibrary.UserDefinedFunctions.Add(udf);

ruleAppDef.UdfLibraries.Add(udfLibrary);

// Add an execute method action to execute UDF
RuleSetDef rs1 = new RuleSetDef("r1");
ExecuteMethodActionDef execUdf = new ExecuteMethodActionDef();
execUdf.AliasName = "StringFunctionsLibrary";
execUdf.MethodName = "IsPatternMatch";
ExecuteMethodActionParamDef p1 = new ExecuteMethodActionParamDef("abcdeedef",
"InputText");
execUdf.ParameterValues.Add(p1);
ExecuteMethodActionParamDef p2 = new ExecuteMethodActionParamDef("deed",
"Pattern");
execUdf.ParameterValues.Add(p2);

rs1.Rules.Add(execUdf);
entity1.RuleElements.Add(rs1);
```

6.2.1.7 Inline table and Value lists access

Namespaces: InRule.Repository, InRule.Repository.RuleElements,
Classes: RuleApplicationDef, TableDef
See Also:
References: InRule.Repository.dll, InRule.Common.dll

There are changes to the techniques used to access inline tables; in InRule SDK 4.5 and later these are accessed through the RuleAppDef see below:

```
RuleApplicationDef ruleAppDef = RuleApplicationDef.Load(@"c:\temp\test.ruleapp");

// Loop through all the data elements
foreach (var dataDef in ruleAppDef.DataElements)
{
    // try to cast the dataDef into a tableDef - if this fails
    // then we will ignore the dataDef since we will only process tables
    TableDef tableDef = dataDef as TableDef;
    if (tableDef != null)
    {
        // Check its an inline table - will not process linked tables
        if (tableDef.TableSettings.TableSourceType == TableSourceType.Inline)
        {
            // get the name
            string tableName = tableDef.Name;

            // Get the data - this adds all the cols in the table
            List<string> columnNames = new List<string>();
            foreach (var col in tableDef.TableSettings.InlineDataTable.Columns)
            {
                columnNames.Add(col.ToString());
            }

            // Iterate the rows in the table and rows to XML
            foreach (System.Data.DataRow dr in
tableDef.TableSettings.InlineDataTable.Rows)
            {
                // only access the fields in the collection of names- you could
                // filter using this.
                foreach (string colName in columnNames)
                {
                    // get the data for each column name
                    string rowItemData = "";
                    rowItemData = string.Format("Name:{0} Value:{1}", colName,
dr[colName].ToString());
                }


            }
        }

    }
}
```

6.2.2 Working with Advanced Definition Objects

Working with Advanced Definition Objects

- Removing Templates in the Language Editor
- Determine FieldDef Dependencies

6.2.2.1 Removing Templates in the Language Editor

Prerequisites: A valid RuleSession, A valid Entity
Namespaces: InRule.Repository, InRule.Repository.Vocabulary, InRule.Repository.Templates
Classes: RuleApplicationDef, TemplateDef
See Also: Embedding the Language Rule Editor
References: InRule.Repository.dll

The following example demonstrates how to remove Templates from the Business Language Editor. This code must run before the Populate method for the Business Language Editor is called.

Removing built in InRule Templates

```
// Create a new VocabularyDef if Vocabulary is null
```

```
if (ruleAppDef.Vocabulary == null)
{
   ruleAppDef.Vocabulary = new VocabularyDef();
}

// Remove the "Minus" template by adding a TemplateAvailabilitySettings and
setting availability to Exclude
ruleAppDef.Vocabulary.TemplateAvailabilitySettings.Add(TemplateLiterals
.TemplateGuids.Minus,
   InRule.Repository.Vocabulary.TemplateAvailability.Exclude);
```

Removing Rule Templates

```
// Make sure the Vocabulary is not null
if (ruleAppDef.Vocabulary == null)
{
    // Create a new VocabularyDef if null
    ruleAppDef.Vocabulary = new VocabularyDef();
}
// Get rule template by name (Note: this is not the same as the Display Name)
TemplateDef templateDef = (TemplateDef)
ruleAppDef.Vocabulary.Templates["ExpressionTemplateDef1"];

// Remove the template using it's GUID
ruleAppDef.Vocabulary.TemplateAvailabilitySettings.Add(templateDef.Guid,
    TemplateAvailability.Exclude);
```

6.2.2.2 Determine FieldDef Dependencies

Prerequisites: A valid RuleRepositoryDefBase, A valid EvalNetwork
Namespaces: InRule.Repository, InRule.Repository.Infos
Classes: RuleApplicationDef, RuleRepositoryDefBase, DefUsageNetwork, FieldDef
References: InRule.Common.dll, InRule.Repository.dll

Determine FieldDef Dependencies

The following recursive method shows how to identify all FieldDef dependencies for a given RuleRepositoryDefBase object.

```
public static IEnumerable<DefUsage> GetDependentFieldDefs(RuleRepositoryDefBase
def, DefUsageNetwork network)
{
   // Create collection of all usages by the given definition
   var usages = network.GetDefUsages(def.Guid, true);

   // Return list of dependencies
   return usages;
}
```

Instantiate the DefUsageNetwork

The following code snippet demonstrates the proper way to create an instance of the DefUsageNetwork.

```
public static DefUsageNetwork GetDefUsageNetwork(RuleApplicationDef ruleAppDef)
{
   return DefUsageNetwork.Create(ruleAppDef);
}
```

6.2.3 Working with the Regression Tester SDK

Working with the Regression Tester SDK

- Authoring a Simple Test Suite for Regression Testing

6.2.3.1 Authoring a Simple Test Suite for Regression Testing

Prerequisites: none
Namespaces: InRule.Repository, InRule.Repository.Regression
Classes: RuleApplicationDef, EntityDef, FieldDef, RuleSetDef, SimpleRuleDef, FireNotificationActionDef, SetValueActionDef, AddCollectionMemberActionDef, TestSuiteDef, DataStateDef, TestContextDef, DataStateMappingDef, ZipFileTestSuitePersistenceProvider
References: InRule.Repository.dll,

The following illustrates the creation of a Rule Application and Test Suite with irSDK. Both are saved as files to the C:\Temp directory. This directory should be created with write privileges before testing. The test suite generated by this sample can be used with the sample code for Executing a Simple Test Suite.

The Rule Application below performs 4 different actions if Field1 is greater than zero. Although Field1 is a String field, the rule uses implicit casting to compare it as an integer. The RuleSet is set to use the Auto fire mode, so ApplyRules is used to execute this rule. The Test Suite creation in the second method illustrates how the Test and Data State are set up, and also how the Assertions are authored to validate the 4 rules that fire, along with checking that the IfThen rule fired. The important points to note are:

- The RuleApplicationDef is assigned to the TestSuiteDef right after creating it. This should always be one of the first actions performed.
- The DataStateDef is assigned to both the TestSuiteDef.RootDataFolder and the TestDef's DataStates collection.
- The TestDef is created by passing a TestContextDef instance to the constructor that was created from the EntityDef of the root entity.
- In this case, the TestDef is added to the TestSuiteDef.RootTestFolder, but it could have been added to a FolderDef hierarchy that we will see in the next example.
- The Assertions use object-relational path notation to address the Rule Application schema, as opposed to XPath notation.
- Not all Assertions require an ExpectedValue and ExpectedValueType.

The files generated from these two methods can be loaded into irAuthor and irVerify:

1. Load C:\Temp\SimpleRuleApp.ruleapp into irAuthor.
2. Launch irVerify from the Entity1 context.
3. Use the File menu to load the C:\Temp\SimpleTestSuite.testsuite file.
4. By clicking on the Test1 tree node, the 5 Assertions will be visible in the Business Language editor.
5. The Test control should show that DataState1 is being used, and that ApplyRules will be performed.
6. By clicking the 'Run All' button, the tests will execute, and all Assertions should pass.
7. By clicking the 'Edit' button next to the Data State drop-down menu on the Test control, the initial state that was stored in the Data State may be viewed, edited and tested in a familiar irVerify style window.

```
public static RuleApplicationDef CreateTestingRuleApp()
```

```
{
    RuleApplicationDef ruleApp = new RuleApplicationDef("SetValueTest");
    EntityDef entity = new EntityDef("Entity1");
    ruleApp.Entities.Add(entity);
    FieldDef field1 = new FieldDef("Field1", DataType.String);
    FieldDef field2 = new FieldDef("Field2", DataType.Integer);
    FieldDef collection1 = new FieldDef("Collection1", DataType.collection);
    FieldDef field3 = new FieldDef("Field3", DataType.DateTime);
    collection1.FieldDefType = FieldDefType.Collection;
    collection1.Fields.Add(field3);
    entity.Fields.Add(field1);
    entity.Fields.Add(field2);
    entity.Fields.Add(collection1);
    RuleSetDef ruleSet = new RuleSetDef("RuleSet1");
    ruleSet.FireMode = RuleSetFireMode.Auto;
    ruleSet.RunMode = RuleSetRunMode.Sequential;
    entity.RuleElements.Add(ruleSet);
    SimpleRuleDef ifthen = new SimpleRuleDef("IfThen1", "Field1 > 0");
    ifthen.SubRules.Add(new SetValueActionDef("Field2", "Field1 + 5"));
    ifthen.SubRules.Add(new AddCollectionMemberActionDef("Collection1"));
    ifthen.SubRules.Add(new SetValueActionDef("Collection1(1).Field3", "Now()"));
    FireNotificationActionDef notification = new
FireNotificationActionDef("Notification1");
    notification.NotificationType = NotificationType.Informational;
    notification.NotificationMessageText = "Test Notification";
    ifthen.SubRules.Add(notification);
    ruleSet.Rules.Add(ifthen);

    ruleApp.SaveToFile(@"C:\Temp\SimpleRuleApp.ruleapp");

    return (ruleApp);
}

public static TestSuiteDef WriteSimpleTestSuite()
{
    // Create a new Test Suite
    TestSuiteDef suite = TestSuiteDef.Create();
    suite.Settings.Name = "SimpleTestSuite";
    suite.ActiveRuleApplicationDef = CreateTestingRuleApp();

    // Create an EntityState Data State with the initial state
    DataStateDef dataState = new DataStateDef();
    dataState.DisplayName = "DataState1";
    dataState.RootEntityName = "Entity1";
    dataState.DataStateType = DataStateType.EntityState;
    dataState.StateXml = "<Entity1><Field1>2</Field1></Entity1>";
    suite.RootDataFolder.Members.Add(dataState);

    // Create a Test and set the Data State as the root mapping
    EntityDef entity1 = suite.ActiveRuleApplicationDef.Entities["Entity1"];
    TestDef test = new TestDef(TestContextDef.Create(entity1));
    test.DisplayName = "Test1";
    test.DataStates.Add(new DataStateMappingDef(dataState));
    suite.RootTestsFolder.Members.Add(test);
    // Create an Assertion to check that Field2 equals 7
    AssertionDef assertion = new AssertionDef();
    assertion.AssertionType = AssertionType.FieldIsEqualToX;
    assertion.TargetElementPath = "Entity1.Field2";
    assertion.ExpectedValue = "7";
    assertion.ExpectedValueType = ExpectedValueDataType.Integer;
    test.Assertions.Add(assertion);

    // Create an Assertion to check that there is 1 collection memeber
    assertion = new AssertionDef();
    assertion.AssertionType = AssertionType.CollectionCountIsX;
    assertion.TargetElementPath = "Entity1.Collection1";
    assertion.ExpectedValue = "1";
    assertion.ExpectedValueType = ExpectedValueDataType.Integer;
    test.Assertions.Add(assertion);

    // Create an Assertion to check the DateTime in Field3
    assertion = new AssertionDef();
    assertion.AssertionType = AssertionType.FieldIsAfterX;
    assertion.TargetElementPath = "Entity1.Collection1(1).Field3";
    assertion.ExpectedValue = "#01/01/2000 00:00#";
    assertion.ExpectedValueType = ExpectedValueDataType.DateTime;
    test.Assertions.Add(assertion);

    // Create an Assertion to check a Notification fired
    assertion = new AssertionDef();
    assertion.AssertionType = AssertionType.NotificationFired;
    assertion.TargetElementPath = "Entity1.RuleSet1.IfThen1.Notification1";
    test.Assertions.Add(assertion);

    // Create an Assertion to check that the IfThen rule fired
    assertion = new AssertionDef();
    assertion.AssertionType = AssertionType.RuleFired;
    assertion.TargetElementPath = "Entity1.RuleSet1.IfThen1";
    test.Assertions.Add(assertion);

    // Persist Test Suite to file system
    suite.SaveAs(new ZipFileTestSuitePersistenceProvider(@"C:\Temp
\SimpleTestSuite.testsuite"));

    return (suite);
}
```

6.2.4 Embedding Authoring Controls

Embedding Authoring Controls

- Embedding InRule Default Editors
- Embedding the Language Rule Editor

- Embedding the Decision Table Editor
- Embedding the Condition Editor

6.2.4.1 Embedding InRule Default Editors

Namespaces: InRule.Repository, InRule.Repository.RuleElements, InRule.Authoring.Controls, InRule.Authoring.Editors, InRule.Authoring.Editors.Controls
Classes: RuleApplicationDef, ControlFactory
See Also: Opening a RuleApplicationDef for Authoring, WinForm Considerations
References: InRule.Repository.dll, InRule.Authoring.dll, InRule.Authoring.BusinessLanguage.dll, InRule.Authoring.Editors.dll, InRule.Common.dll

Loading the default editors

The following code works in WPF. This same code can be used for retrieving all editors as the GetControl method takes a RuleRepositoryDefBase type. The editor that is returned is based on the type of definition object that is passed in. In this example, the Fire Notification Action is used.

```
// Create an instance of the control factory (consider keeping this at form level
to reuse where appropriate)
ControlFactory controlFactory = new ControlFactory();

// Load ruleapplicationdef
RuleApplicationDef ruleAppDef = RuleApplicationDef.Load(@"c:\work
\mortgagecalculator.ruleapp");

// Load the rule application into the control factory
controlFactory.OpenRuleApplication(ruleAppDef);

// Get the entity
EntityDef entityDef = ruleAppDef.Entities["Mortgage"];

// Get the rule to edit (any RuleRepositoryDefBase can be passed in)
FireNotificationActionDef ruleDef = (FireNotificationActionDef)
entityDef.GetRuleSet("PaymentRules").Rules["PaymentNotification"];

// Get the business language editor and load into ContentControl
var control = controlFactory.GetControl(ruleAppDef);
```

Saving the changes

```
// Get the control from the form
if (control is IValidatingEditor)
{
    // Save the changes back to the rule application
    ((IValidatingEditor)control).SaveValues();
}

// Dispose of the control if it implements IDisposable
if (control is IDisposable)
{
    ((IDisposable)control).Dispose();
}

// Save the rule application back to the file system (this can also be saved to the
Catalog)
ruleAppDef.SaveToFile(@"c:\work\mortgagecalculator.ruleapp");
```

6.2.4.2 Embedding the Language Rule Editor

Namespaces: InRule.Repository, InRule.Repository.RuleElements, InRule.Authoring.BusinessLanguage, InRule.Authoring.Controls, InRule.Authoring.Editors, InRule.Authoring.Editors.Controls
Classes: RuleApplicationDef, LanguageRuleDef, BusinessLanguageEditor, ControlFactory
See Also: Opening a RuleApplicationDef for Authoring, WinForm Considerations
References: InRule.Repository.dll, InRule.Authoring.dll, InRule.Authoring.BusinessLanguage.dll, InRule.Authoring.Editors.dll, InRule.Common.dll

Loading the business language editor into a ContentControl

The following code works in WPF. Loading the control in this manner will only load the control, the Name and Enabled fields will not be present. To embed the entire control see here.

```
// Create an instance of the control factory (consider keeping this at form level
to reuse where appropriate)
ControlFactory controlFactory = new ControlFactory();

// Load ruleapplicationdef
RuleApplicationDef ruleAppDef = RuleApplicationDef.Load(@"c:\work
\mortgagecalculator.ruleapp");

// Load the rule application into the control factory
controlFactory.OpenRuleApplication(ruleAppDef);

// Get the entity
EntityDef entityDef = ruleAppDef.Entities["Mortgage"];

// Get the language rule to edit
LanguageRuleDef ruleDef = (LanguageRuleDef)
entityDef.GetRuleSet("PaymentRules").Rules["CalcPaymentSummary"];

// Get the business language editor and load into ContentControl
var control = controlFactory.CreateBusinessLanguageEditor(ruleDef);
```

Saving the changes

```
// Get the control from the form
if (control is BusinessLanguageEditor)
{
    // Save the changes back to the rule application
    control.Save();
}

// Dispose of the control if it implements IDisposable
if (control is IDisposable)
{
    control.Dispose();
}

// Save the rule application back to the file system (this can also be saved to the
Catalog)
ruleAppDef.SaveToFile(@"c:\work\mortgagecalculator.ruleapp");
```

6.2.4.3 Embedding the Decision Table Editor

Namespaces: InRule.Repository, InRule.Repository.RuleElements, InRule.Authoring.BusinessLanguage, InRule.Authoring.Controls, InRule.Authoring.Editors, InRule.Authoring.Editors.Controls, InRule.Repository.DecisionTables
Classes: RuleApplicationDef, DecisionTableControl, ControlFactory
See Also: Opening a RuleApplicationDef for Authoring, WinForm Considerations
References: InRule.Repository.dll, InRule.Authoring.dll, InRule.Authoring.BusinessLanguage.dll, InRule.Authoring.Editors.dll, InRule.Common.dll

Loading the decision table editor into a ContentControl

The following code works in WPF. Loading the control in this manner will only load the control, the Name and Enabled fields will not be present. To embed the entire control see here.

```
// Create an instance of the control factory (consider keeping this at form level
to reuse where appropriate)
ControlFactory controlFactory = new ControlFactory();

// Load ruleapplicationdef
RuleApplicationDef ruleAppDef = RuleApplicationDef.Load(@"c:\work
\invoice.ruleapp");

// Get the entity
EntityDef entityDef = ruleAppDef.Entities["Mortgage"];

// Get the decision table to edit
DecisionTableDef decisionTableDef = (DecisionTableDef)
entityDef.GetRuleSet("InvoiceRules").Rules["CommissionTable"];

// Load decision table editor into content control
var control = controlFactory.CreateDecisionTableControl(decisionTableDef);
```

Saving the changes

```
// Changes from the decision table control are automatically saved back to the rule
app
// So we are just going to save rule app to file system (or Catalog if desired)
ruleAppDef.SaveToFile(@"c:\work\invoice.ruleapp");
```

6.2.4.4 Embedding the Condition Editor

Namespaces: InRule.Repository, InRule.Repository.RuleElements, InRule.Authoring.BusinessLanguage, InRule.Authoring.Controls, InRule.Authoring.Editors, InRule.Authoring.Editors.Controls
Classes: RuleApplicationDef, ControlFactory
See Also: Opening a RuleApplicationDef for Authoring, WinForm Considerations
References: InRule.Repository.dll, InRule.Authoring.dll, InRule.Authoring.BusinessLanguage.dll, InRule.Authoring.Editors.dll, InRule.Common.dll

Loading the condition editor into a ContentControl

The following code works in WPF. Loading the control in this manner will only load the control, the Name and Enabled fields will not be present. To embed the entire control see here.

```
// Create an instance of the control factory (consider keeping this at form level
to reuse where appropriate)
ControlFactory controlFactory = new ControlFactory();

// Load ruleapplicationdef
RuleApplicationDef ruleAppDef = RuleApplicationDef.Load(@"c:\work
\mortgagecalculator.ruleapp");

// Get the entity
EntityDef entityDef = ruleAppDef.Entities["Mortgage"];

// Get the decision table to edit
SimpleRuleDef simpleRuleDef = (SimpleRuleDef)
entityDef.GetRuleSet("CalcPaymentSchedule").Rules["IfLoanIsValid"];

// Load condition editor into content control, defaulting to business language mode
(false for last parameter will load in syntax mode)
var control  = controlFactory.CreateConditionEditor(simpleRuleDef,
    simpleRuleDef.Condition.FormulaText,
    true);
```

Saving the changes

```
if (control != null)
{
    // Call save values
    control.SaveValues();

    // If user is in syntax mode, then we need to manually save the changes back to
the rule
    if (!control.UseLanguageExpression)
    {
            // Get the rule from the editor
            var contextDef = (SimpleRuleDef)control.ContextDef;

            // Update the definition object
            contextDef.ConditionExpression = control.Expression;
    }
}
```

6.3 Catalog API Examples

The InRule Catalog API demonstrates how to retrieve and work with rule applications that are stored in the InRule Catalog.

Developing with rule applications stored in the Catalog

- Creating a RuleCatalogConnection
- Opening a RuleApplicationDef for Authoring from Catalog
- Working with RuleApplicationDef in the Catalog
- Working with Rule Elements in the Catalog
- Sharing Elements in the Catalog
- Performing Rollbacks in the Catalog

- Promote a Rule Application from Catalog to Catalog

6.3.1 Creating a RuleCatalogConnection

Prerequisites: None
Namespaces: InRule.Repository.Client
Classes: RuleCatalogConnection

The rule catalog connection object is used to obtain the RuleApplicationDef from the catalog for authoring rules in code.

Creating a Catalog connection with URI

```
// Create a RuleCatalogConnection passing in the catalog URI
RuleCatalogConnection connection = new RuleCatalogConnection(new
Uri(repositoryUri));
```

Creating a Catalog connection with credentials

```
// Create a RuleCatalogConnection passing in the catalog URI and login credentials
RuleCatalogConnection connection = new RuleCatalogConnection(new
Uri(repositoryUri),new TimeSpan(0,10,0),"Admin","password");
```

Getting a Catalog connection with oAuthManager

```
var settings = new OAuthSettings();
settings.Audience = "YOUR_VALUE";
settings.ClientId = "YOUR_VALUE";
settings.Authority = "YOUR_VALUE";
settings.RedirectUri = "YOUR_VALUE";
settings.LogoutUrl = "YOUR_VALUE";

var oAuthManager = new OAuthManager(settings);

// Gets a RuleCatalogConnection from your oAuthManager instance
RuleCatalogConnection connection =
 oAuthManager.CreateCatalogConnectionAsync(repositoryUri, new TimeSpan(0,10,0),
false);
```

6.3.2 Opening a RuleApplicationDef for Authoring from Catalog

Prerequisites: None
Namespaces: InRule.Repository
Classes: RuleApplicationDef, RuleAppRef, RuleCatalogConnection
See Also: Creating a RuleCatalogConnection, Opening a Rule ApplicationDef for Authoring

The RuleApplicationDef object is used to dynamically create and modify rules and schema elements in code.

Load a specific revision of a rule application from the catalog

```
// Get a RuleAppRef object which will contain details about the rule app we are
```

```
loading
RuleAppRef ruleAppRef = connection.GetRuleAppRef("InvoiceApp");

// Load the RuleApplicationDef from the repository, specifying the GUID and
Revision
// This can be retrieved by name or GUID, with option to specify label and revision
RuleApplicationDef ruleAppDef =
connection.GetSpecificRuleAppRevision(ruleAppRef.Guid, ruleAppRef.PublicRevision);
```

Load the latest revision of a rule application from the catalog

```
// Load the RuleApplicationDef from the repository, specifying the name
RuleApplicationDef ruleAppDef = connection.GetLatestRuleAppRevision("RuleAppName");
```

6.3.3 Working with RuleApplicationDef in the Catalog

Prerequisites: A valid RuleCatalogConnection and RuleApplicationDef object
Namespaces: InRule.Repository, InRule.Repository.Client, InRule.Repository.Service.Data
Classes: RuleCatalogConnection, RuleApplicationDef

The following examples demonstrate how to save, check-in and check-out rule applications and their granular elements from the Catalog service.

Create new rule application in the Catalog

```
// save a new rule application to the Catalog
connection.CreateRuleApplication(ruleAppDef, "New app description");
```

Save a rule application in the Catalog without checking in

```
// Save pending changes without checking them in
connection.Save(ruleAppDef);
```

Note: The rule application is stored in a user specific location and will allow saving even if validation errors exist.

Check out a rule application from the Catalog

```
// Check out the rule application
connection.CheckoutRuleApplication(ruleAppDef, false, "comments");
```

Check out a rule application from the Catalog along with the Schema

```
// Check out the rule application and the schema
connection.CheckoutRuleApplicationAndSchema(ruleAppDef, "comments");
```

Note: The rule application must be checked out in order to check out the schema.

Check in a rule application to the Catalog

```
// Check in pending changes
// This will check in all pending changes on the rule app
connection.Checkin(ruleAppDef, "comments");
```

Undoing a checkout for a rule application in the Catalog

```
// Undo the check out of the rule app
// This will undo all checked out elements of the rule application
connection.UndoRuleAppCheckout(ruleAppDef);
```

Determine if a rule application exists in the Catalog

```
// This will return a boolean stating whether or not the rule application exists in
```

```
the Catalog by name
// Overloads exist for name, name and revision, name and label
connection.DoesRuleAppExist("MortgageCalculator");
```

6.3.4 Working with Rule Elements in the Catalog

Prerequisites: A valid RuleCatalogConnection and RuleApplicationDef object
Namespaces: InRule.Repository, InRule.Repository.Client, InRule.Repository.Service.Data
Classes: RuleCatalogConnection, RuleApplicationDef, EntityDef, RuleRepositoryDefBase
See Also: Working with RuleApplicationDef in the Catalog

The following examples demonstrate how to check-out rule application elements from the Catalog.

Note: When checking in or saving an element, the entire RuleApplicationDef is checked in or saved, not just the element that you are working with. See Working with RuleApplicationDef in the Catalog to see how to perform these actions.

Check out a RuleSet from the Catalog

```
// Get EntityDef
EntityDef entityDef = ruleAppDef.Entities["Invoice"];

// Get the RuleSet off of the EntityDef
RuleRepositoryDefBase ruleSetDef = entityDef.GetRuleSet("DiscountRules");

// Check out the RuleSet
connection.CheckoutDef(ruleAppDef, ruleSetDef, "comments");
```

Check out a DataElement from the Catalog

```
// Get the DataElement (a table in this case)
RuleRepositoryDefBase table = ruleAppDef.DataElements["Customer"];

// Check out the DataElement
connection.CheckoutDef(ruleAppDef, table, "comments");
```

6.3.5 Sharing Elements in the Catalog

Prerequisites: A valid RuleCatalogConnection and RuleApplicationDef object
Namespaces: InRule.Repository, InRule.Repository.Client, InRule.Repository.Service.Data
Classes: RuleCatalogConnection, RuleApplicationDef, EntityDef, RuleRepositoryDefBase
See Also: Working with RuleApplicationDef in the Catalog

The following examples demonstrate how to share and consume rule elements in the Catalog.

Sharing a rule set

```
// set ruleset shareable
catalogConn.SetShareableFlagForDef(ruleAppDef.Entities["Entity1"].RuleElements["Rul
eSet1"].Guid, true);
```

Consuming a shared rule set in an existing rule application

```
// Get source rule app
RuleApplicationDef sharedRuleApp =
catalogConn.GetLatestRuleAppRevision("SharedRuleApp");

// Get rule set to consume from source rule app
```

```
RuleSetDef ruleSetDef = sharedRuleApp.Entities["Entity1"].GetRuleSet("RuleSet1");

// Get consuming rule app
RuleApplicationDef consumingRuleApp =
catalogConn.GetLatestRuleAppRevision("ConsumingRuleApp");

// Check out the rule app container
catalogConn.CheckoutRuleApplication(consumingRuleApp, false, "checkout comments");

// Cet entity where rule set will be added
EntityDef entityDef = consumingRuleApp.Entities["Entity1"];

// Add shared rule set, using same Guids will create the share
entityDef.RuleElements.Add(ruleSetDef.CopyWithSameGuids());

// Check in the rule application
catalogConn.Checkin(consumingRuleApp, "check in comments");
```

Sharing a schema

```
// Share the schema so it can be consumed by other rule applications
catalogConn.SetShareableFlagForDef(sharedRuleApp.SchemaGuid, true);
```

Binding the shared schema to a rule application

```
// Bind the rule
catalogConn.SetMasterRuleAppForDef(sharedRuleApp.SchemaGuid, sharedRuleApp.Guid);

// Create new rule application that will consume the shared schema
RuleApplicationDef consumerRuleApp = new RuleApplicationDef("ConsumingRuleApp");

// Add the rule application to the Catalog
catalogConn.CreateRuleApplication(consumerRuleApp, consumerRuleApp.Name);

// Check out the rule application
catalogConn.CheckoutRuleApplication(consumerRuleApp, true, "checkout consumer
ruleapp");

// Get all shared elements in the Catalog
ResultMap<DefInfo, DefInfo> shareableDefs = catalogConn.GetAllShareableDefs();

// Using a Linq query to find the schema we are interested in by name
DefInfo sharedSchemaDefInfo = (
    from d in shareableDefs.Keys
    where d.MasterRuleappName == "SharedRuleApp" &&
d.Key.DefType.Equals(typeof(RuleApplicationSchemaDef))
    select d).First();

// Add the shared schema to the derived rule app
consumerRuleApp = catalogConn.ReplaceRuleAppSchema(consumerRuleApp,
sharedSchemaDefInfo.Key);

// Check-in rule app
catalogConn.Checkin(consumerRuleApp, "Comments");
```

6.3.6 Performing Rollbacks in the Catalog

Prerequisites: A valid RuleCatalogConnection

Namespaces: InRule.Repository, InRule.Repository.Client, InRule.Repository.Service.Data
Classes: RuleCatalogConnection, RuleApplicationDef, RuleAppRef, RuleRepositoryDefBase
See Also: Creating a RuleCatalogConnection

The following examples demonstrate how to perform rollback functionality in the Catalog. The functionality does not perform a true rollback, instead it will save the previously desired rule application or rulse set back to the Catalog as the latest revision.

Rollback to a previous rule application

```
// Get a RuleAppRef object which will contain details about the rule app we are
loading
RuleAppRef ruleAppRefLatest = connection.GetRuleAppRef("Historical");

// Get a RuleAppRef object which will contain details about the rule app we want to
become the latest
RuleAppRef ruleAppRefPrior = connection.GetRuleAppRef("RuleAppRollback", 2);

// Load the RuleApplicationDef from the repository, specifying the GUID and
Revision
RuleApplicationDef ruleAppDefPrior =
connection.GetSpecificRuleAppRevision(ruleAppRefPrior.Guid,
ruleAppRefPrior.PublicRevision);

// overwrite the existing rule app with the prior version
connection.OverwriteRuleApplication(ruleAppRefLatest.Guid, ruleAppDefPrior, true,
"replacing latest with prior version");
#endregion
```

Rollback to a previous rule set

```
// Get a RuleAppRef object which will contain details about the rule app we are
loading
RuleAppRef ruleAppRefLatest = connection.GetRuleAppRef("InvoiceRollbackExample");

// Load the RuleApplicationDef from the repository, specifying the GUID and
Revision
RuleApplicationDef ruleAppDefLatest =
connection.GetSpecificRuleAppRevision(ruleAppRefLatest.Guid,
ruleAppRefLatest.PublicRevision);

// Get the RuleSet
EntityDef entityDefLatest = ruleAppDefLatest.Entities["LineItem"];
RuleRepositoryDefBase ruleSetDefLatest =
entityDefLatest.GetRuleSet("LineItemRules");

// Check out the RuleApp
connection.CheckoutRuleApplication(ruleAppDefLatest, false, "checking out rule
application");

// Get a RuleAppRef object which will contain details about the rule app we are
loading
RuleAppRef ruleAppRefPrior = connection.GetRuleAppRef("InvoiceRollbackExample", 1);

// Load the RuleApplicationDef from the repository, specifying the GUID and
Revision
RuleApplicationDef ruleAppDefPrior =
connection.GetSpecificRuleAppRevision(ruleAppRefPrior.Guid,
ruleAppRefPrior.PublicRevision);

// Get the RuleSet
```

```
EntityDef entityDefPrior = ruleAppDefPrior.Entities["LineItem"];
RuleRepositoryDefBase ruleSetDefPrior = entityDefPrior.GetRuleSet("LineItemRules");

// Remove the ruleset from the latest revision
entityDefLatest.RuleElements.Remove(ruleSetDefLatest);

// Add prior revision of the ruleset to the latest entity
entityDefLatest.RuleElements.Add(ruleSetDefPrior.Copy());

// Checkin the rule app
connection.Checkin(ruleAppDefLatest, "rolled back to prior revision of
LineItemRules");
```

6.3.7 Promote a Rule Application from Catalog to Catalog

Prerequisites: Valid RuleCatalogConnection to both Source and Target Catalog.
Namespaces: InRule.Repository
Classes: RuleApplicationDef, RuleAppRef, RuleCatalogConnection
See Also: Creating a RuleCatalogConnection

```
// Get a reference to the rule application from the source Catalog (doing by name
here, label and revision could be used as well)
RuleAppRef ruleAppRef = sourceConn.GetRuleAppRef("AutoDepreciation");

// Get the rule application from the Source Catalog
RuleApplicationDef sourceRuleAppDef =
sourceConn.GetSpecificRuleAppRevision(ruleAppRef.Guid, ruleAppRef.PublicRevision);

// Promote the rule application to the Target Catalog
RuleApplicationDef targetRuleAppDef =
targetConn.PromoteRuleApplication(sourceRuleAppDef, "Description", "Comments");
```

Part VII

InRule Samples

7 InRule Samples

InRule provides numerous samples for learning how to use InRule to its fullest potential. The samples show how to use InRule within an industry application or demonstrate innovative ways to apply InRule to specific logic/programming problems.

You can easily browse the sample library on GitHub. To access the samples, launch irAuthor and select View Samples on GitHub from the main page. Alternatively, you can visit http://samples.inrule.com, which will redirect you to the GitHub repository.

Part VIII

Regression Testing

8 Regression Testing

The InRule Regression Testing feature in irVerify provides a state assertion mechanism similar to NUnit to test state values after rule execution.

While the user interface for Regression Testing in irVerify provides a rich test authoring experience using the Business Language editor, the Regression Testing SDK is designed to be used for programmatic creation of tests, and their execution. This will be a useful feature for automated software builds to validate the consistency of Rule Applications and their input state.

The Regression Testing user interface is built upon the same API that is exposed in the Regression Testing SDK; therefore any tests authored in the user interface may also be executed via the SDK and vice-versa.

See the following topics for detailed information on the Regression Testing SDK:

- Regression Testing Concepts
- Authoring Test Suites
- TestScenario versus EntityState Data States
- Executing Test Suites

Outside of the interface in irVerify and irSDK, there is still one more way with which to perform regression testing with InRule:

- Command Line Interface

8.1 Regression Testing Concepts

Test Suite

All authored tests are contained within a Test Suite. A Test Suite provides a programmatic interface for the adding, removing and finding of tests contained within the suite, provides some test execution settings and sets up the association with a Rule Application.

A persistence provider can be used with the Test Suite to load and save it from a data store.

The NUnit analog of a test suite is a unit test assembly.

Data States

Data States represent the initial state of a test, prior to rule execution. The XML data contained within them is loaded into their associated Entities when the tests are executed. Data States may either represent an irVerify TestScenario (serialized RuleSessionState) or a discrete EntityState.

Individual Data States may be shared by one or more Tests.

Tests

Tests represent a collection of Assertions. They act on a specific context such as an Entity or Independent RuleSet. They govern whether the RuleSession during Test execution will either ApplyRules, or execute an explicit RuleSet.

If more than one Data State is used by the test, it also handles the mapping of the Data States to temporary linked Entity fields or Independent RuleSet’s Entity arguments.

There is always a new RuleSession and RuleServiceConnection created for each Test’s execution. The following procedure occurs during Test execution:

- A new RuleSession and RuleServiceConnection are created for the test, setting up any EndPoint Overrides that may exist.

- Initial state is loaded into Entities or Independent RuleSets from the Data States.
- The RuleSession calls ApplyRules or executes an explicit RuleSet.
- The Assertions are processed against the final state, producing results.

The NUnit analog of a Regression Testing Test is a unit test.

Assertions

Assertions validate individual output state values from fields, calculations, collections, notifications or rule elements against expected values asserted at authoring time. There are many different types of Assertion to choose from.
When they are executed, they generate either a pass or fail, and highlight their expected and actual values. For a Test to pass, all of its constituent Assertions must also pass.
The NUnit analog of an Assertion is the Assert statement.

Folders

Folders act as both a user interface feature and a means to group common Tests together. While folders may contain any Regression Testing construct, they are primarily used to provide a hierarchical grouping for Tests.

All Data States are contained within a root Data State folder on the Test Suite. Although an SDK author may use folders to group Data States hierarchically, this is prohibited in the user interface as it serves no purpose.

There is also a root Test folder on the Test Suite. This may contain a mixture of Tests and other Folders which may contain their own Folders and Tests.

Tests may be executed at the folder level so that only a specific group of Tests is executed.

The NUnit analog of a Folder is a TestFixture.

Data State Overrides

These provide the ability to override individual state values at Test execution time. By applying them to a Test, the value of a single field may be modified, which overrides its original input state defined in the Test's Data State.

This feature may be useful if some of the Rules change in a Rule Application, and the Test's DataState requires fine-grained adjustment in order to satisfy the Test's Assertions.

8.2 Authoring Test Suites

The classes required for Test Suite authoring are located in the InRule.Repository assembly beneath the InRule.Repository.Regression namespace.

See Authoring a Simple Test Suite for Regression Testing for a sample

Most of the authoring Def classes inherit from the IdentifiedDefBase class, which simply provides them with a GUID identifier to assist persistence. The DefBase also provides them with the following:

- Container property - Reference to the FolderDef container.
- Parent property - Reference to the parent collection or Def.
- GetAncestor<T> method - The first ancestor in the parent hierarchy matching the type T.

It is unlikely you will need to use these members for regular Test Suite authoring. They are provided, along with the INotifyPropertyChanged interface, to assist the user interface.

API Documentation

Most members of each class are self-explanatory, so only the members of interest will be mentioned:

TestSuiteDef

- ActiveRuleApplicationDef - Use this property to associate a RuleApplicationDef with the Test Suite. This should be the first step performed after creating the Test Suite. The setter contains validation logic against existing Tests and Data States in the Test Suite so an InvalidRuleAppForTestSuiteException will be thrown containing the GUIDs of invalid Tests and Data States if the Rule Application’s schema does not match up. Also, a MismatchedTestSuiteRuleAppGuidException will be thrown if the Rule Application’s GUID identifier does not match up with the one used when then Test Suite was authored.
- RootDataFolder - Use this property to add each new DataStateDef to the TestSuiteDef. This should be performed in addition to assigning the DataStateDef to a TestDef.
- RootTestFolder - Use this property to add each new TestDef or FolderDef to the TestSuiteDef.
- Settings - Use this property to access the TestSuiteSettingsDef which should always exist for a TestSuiteDef.
- Create() - Use this method to create a new empty TestSuiteDef instance. There is no public constructor on the TestSuiteDef to prohibit direct XML serialization; use the SaveAs() and Save() methods to achieve serialization via a persistence provider.
- LoadFrom() - Use this static method to load an existing TestSuiteDef from the specified persistence provider.
- Save() - Use this method to save the current TestSuiteDef with the persistence provider previously set using the SetPersistenceProvider() method.
- SaveAs() - Use this method to save the current TestSuiteDef with the specified persistence provider.

TestSuiteSettingsDef

There is a 1:1 relationship between the TestSuiteDef and the TestSuiteSettingsDef. It represents the TestSuiteDef during serialization and contains settings for test execution, information Rule Application data, and the origin of a FileSystem or Catalog Rule Application when using the stand-alone version of the user interface.

- IncludeRuleExecutionLog - Use this property to indicate whether the test results should contain the RuleExecutionLog that was generated by the RuleSession when the test executed.
- IncludeXmlRuleTrace - Use this property to indicate whether the test results should contain the XML Rule Trace that was generated by the RuleSession when the test executed. This should be used in conjunction with the TraceFrameTypes property on the RuleExecutionSettings.

TestDef

- Assertions - Use this collection to add AssertionDefs to the Test.
- DataStates - Use this collection to add DataStateMappingDefs to the Test. The mappings connect EntityState Data States together to form a RuleSessionState. If there is only 1 EntityState Data State, or a TestScenario Data State, this will be identified as the “-Root-‟mapping.
- IsEnabled - The test execution will skip this test if this property is set to true.
- Overrides - Use this collection to apply DataStateOverrideDef instances to the test. This allows either an individual field value to be changed in the initial state, or a collection member to be added.
- RootContext - Use this property to set the TestContextDef instance which represents the context (Entity or Independent RuleSet) on which the test will execute. It is more common to pass this into the constructor of the TestDef.

TestContextDef

An instance of a TestContextDef should be generated from the static Create() factory method, rather than instantiating it with its constructor and settings all of its properties.

- Create() - Use this static method to create a new instance of the TestContextDef. Its overloads allow either an EntityDef or RuleSetDef to be passed in, depending on the context required.
- RootContextName - Use this property to set the root context on which the test should execute. This should match the RootEntityName of the root DataStateMappingDef’s

DataStateDef. The Create() method should automatically populate this.

- ExecutionType - Use this property to set the rule execution type (ApplyRules, Execute Explicit RuleSet, Execute Independent RuleSet).
- ExecuteRuleSetName - Use this property to set the name of the RuleSet to execute. (Does not apply to ApplyRules execution type)

DataStateDef

- DataStateType - Use this property to set the type of Data State: TestScenario or EntityState.
- RootEntityName - This should be set to the root Entity name that this DataStateDef represents. If the DataStateType is TestScenario and an Independent RuleSet is the root, then use the name of the Independent RuleSet.
- RootEntityStateId - This should only be set when the DataStateType is TestScenario. In certain scenarios the EntityStateId of the root Element ID, or Independent RuleSet is not 1. (e.g. Entity1::4) If the DataStateType is EntityState or the TestScenario's root EntityStateId is 1, you can safely leave this property unset, as it defaults to 1.
- StateXml - Use this string property to store the XML of either the TestScenario or the EntityState. The TestScenario XML can be obtained by using the XmlSerializer.Serialize() on a RuleSessionState instance (accessible from RuleSession.State property). EntityState XML can be obtained from the Entity.State.GetXml() method.

DataStateMappingDef

- ArgumentName - Use this property to set the name of either the Independent RuleSet Entity argument, or the name of a root-level temporary linked Entity field.
- DataState - Use this property to set the DataStateDef that should be mapped to the specified argument or field.

FolderDef

- FolderType - Use this property to set the type of folder. (TestFolder or DataFolder). This should always be set to TestFolder as all Data States should exist at the root level of the TestSuiteDef's RootDataFolder.
- Members - Use this collection to assign TestDefs to the FolderDef. During the Add() method, the TestDef's GUID is used to add a ReferenceToDef instance (instead of the TestDef itself) to the underlying collection.

ReferenceToDef

This class is used to represent a TestDef, DataStateDef or FolderDef in the hierarchy beneath a FolderDef. This is because the default persistence provider implementation stores the Defs relationally in separate files (in a zip file) rather than hierarchically in a single file. When iterating the FolderDef.Members collection, use the ReferenceToDef.EnsureReferencedDef() method to access the actual TestDef, DataStateDef or FolderDef. Alternatively, use the ReferenceToDef.ReferencedDefId to pass into the TestSuiteDef.GetTest()/GetDataState()/GetFolder() methods, depending on the IsDataStateDef, IsTestDef, IsFolderDef properties.

DataStateOverrideDef

- DataOverrideType - Use this property to set the type of override. (SetValue or AddCollectionMember)
- TargetPath - Use this property to set the path to the state value to override using object-relational path (dot) notation. (E.g. Entity1.Collection1(1).Field1)
- Value - Use this property to set the value of the override. This does not apply to the AddCollectionMember type.

AssertionDef

- AssertionType - Use this property to set the type of Assertion. The AssertionType enum currently contains 29 different types of Assertion to choose from, e.g. FieldIsEqualToX, CollectionCountIsX.
- ExpectedValue - Use this property to set the string representation of the expected value of the Assertion. Not all Assertions have an expected value, e.g. FieldIsNotNull. The string representation should be in the culture-invariant format. Dates should be stored as "#mm/

dd/yyyy#".

- ExpectedValueType - While the Business Language editor can determine the data type of the field from the Rule Application, SDK users should explicitly set this property to the correct data type.
- TargetElementPath - Use this setting to set the target of the Assertion using object-relation path (dot) notation. (E.g. Entity1.Collection1(1).Field1)

ZipFileTestSuitePersistenceProvider
This is currently the only persistence provider implementation which stores XML serialized TestSuiteSettingsDef, FolderDefs, TestDefs and DataStateDefs in individual files within a zip file. SDK users need only pass the output path to its constructor and pass it to the TestSuiteDef.SaveAs() method.
To implement custom persistence providers, create a derived class that inherits from the TestSuitePersistenceProvider base class.

8.3 TestScenario versus EntityState Data States

EntityState XML has the simplicity of being an easily constructed XML document which may either be written by hand (e.g. <Entity1><Field1>5</Field1><Field2>10</Field2></Entity1>) or XML serialized using end-users' own classes.

TestScenario is a more complicated structure which can only be generated by using XmlSerializer on the RuleSessionState class. (RuleSession.State)

The advantages of the TestScenario are:

- It includes temporary state fields and collections which are used for both temporary state, and also cross-schema linked Entities and Collections.
- It will work with object-bound schemas that only work with the BinaryFormatter and not the XmlSerializer.
- It will support circular references.

EntityState XML may still be used to some degree for temporary fields via a collection of the DataStateMappingDef instances. This is restricted to:

- Root-level temporary linked Entity fields.
- Independent RuleSet Entity arguments.

Note - In most testing situations, it is suitable for the user to utilize testscenarios unless the state data must be consumed or deserialized by a separate application.

8.4 Executing Test Suites

The classes required for Test Suite execution reside in the InRule.Runtime.Testing assembly.

See Executing a Simple Test Suite for a sample

API Documentation

TestingSessionManager
This class handles the RuleSession and RuleServiceConnection generation for each TestDef that is executed. It handles the construction of a RuleSessionState from either a TestScenario or discrete EntityState Data States.

- TestingSessionManager() - The constructor of this class requires an instance of an IConnectionFactory implementation. This ensures that a new RuleServiceConnection is created for each Test execution, and its correct EndPoint overrides are used.

- CreateTestingSession() - This method takes a Runtime RuleApplicationReference parameter. If there is only a RuleApplicationDef available, wrap it in an InMemoryRuleApplicationReference instance. It also must be passed a TestDef parameter in order for the DataState mapping retrieval. There is also an out parameter of rootEntityId. This is the EntityStateId integer identified from the root context of a TestScenario.

RegressionTestingSession
This class is the entry-point for Test execution. Its constructor must be passed an instance of the TestingSessionManager for RuleSession management, and a valid TestSuiteDef to execute against. There is an optional constructor overload that also accepts a Runtime RuleApplicationReference instance which can be used if the TestSuiteDef has no ActiveRuleApplicationDef.

- ExecuteAllTests() - This method will execute all Tests within the Test Suite, returning a collection of Test results.
- ExecuteTests() - This method will execute Tests based on the overload parameters passed in. It will execute: A single folder, a list of folders, a list of tests, or a list of folders and tests, returning a collection of Test results.

TestResultCollection
This class represents an immutable collection of TestResults. It can only be created from one of the RegressionTestingSession’s Execute methods. A list of TestResults can be enumerated from this class. The SaveAs() and LoadFrom() methods can be used to load and save the TestResults into a zip file. In addition to the TestResultCollection.xml file stored within the zip file, the TestSuiteDef files and the RuleApplicationDef.xml are also stored. This will provide an accurate audit trail of which Rule Application was executed against which Test Suite in order to produce the results.

Note: Any instance of this class should be explicitly disposed after use to ensure any temporary files it created are cleaned up.

TestResult
This class gives an overview of the result of all the Assertions executed in the Test. The Passed property will only be true if all the AssertionResults in its collection also passed. Other properties of interest:

- RuntimeErrorMessage - This property will contain the error messages of any errors generated, or exceptions thrown during rule execution for this Test. It may also contain errors generated from the Assertion processing after the rule execution has run.
- RuleExecutionLog - This will be populated with the RuleExecutionLog generated from the ApplyRules() or RuleSet.Execute() when the rules were executed. Its output is optional based on the Boolean setting stored in the TestSuiteSettingsDef.
- XmlRuleTrace - This will contain a StringBuilder instance of the XML version of the RuleTrace generated during rule execution. Its output is optional based on the Boolean setting stored in the TestSuiteSettingsDef, and also if any TraceFrameTypes are specified in the irVerify options. Currently these options are only accessible from the hidden irVerify options here: InRule.Runtime.Testing.UserSettings.CurrentTesterOptions.TraceFrameTypes.

AssertionResult
This class represents the result of each AssertionDef executed in a Test. The ExpectedValue and ActualValue fields will contain the values in culture-invariant format. For culture-specific data, use the FormattedExpectedValue and FormattedActualValue properties.

8.5 Command Line Interface

A command-line interface for executing Regression Testing Test Suites is provided with InRule.

The command-line interface can be used to integrate Regression Testing execution into a build process, for example. The command-line executable is installed with irAuthor and is called irVerify.exe.

Command Line Usage

```
irVerify.exe /s TestSuitePath

        [/n TestName[,TestName2,...,TestNameN]]

        [/f FolderName[,FolderName2,...,FolderNameN]]

        [/r TestResultsPath]

        [/l RuleApplicationLocationURI]

        [/w]
```

/s TestSuitePath	Required parameter. Specifies the path to the Test Suite file to be executed.
/n TestName	Optional parameter. Comma separated (no spaces) list of test case names to be executed.
/f FolderName	Optional parameter. Comma separated (no spaces) list of folder names to be executed.
/r TestResultsPath	Optional parameter. Specifies path to test results file to be saved.
/l RuleApplicationLocationURI	Optional parameter. Specifies the location of the Rule Application to be tested. This is a file path for file based Rule Applications and a URI for irCatalog based Rule Applications. **Note:** This parameter is **required unless** the Test Suite was created via the SDK. If using the SDK, the Rule Application location can be embedded within the Test Suite file.
/w	Optional parameter. Suppresses windows.

Usage Examples

The following examples will be based on a Rule Application called ExampleRuleApp. The rest of the information about this Rule Application is listed in the table below.

/s TestSuitePath	Example.testsuite
/n TestName	ExampleTestCase1, ExampleTestCase2 **Note:** These are tests within Example.testsuite
/f FolderName	ValidationTestFolder and PromotionTestFolder **Note:** These are folders within Example.testsuite that contain tests
/r TestResultsPath	Example.testresults **Note:** This file will be created if this parameter is used. If the file already exists, it will be overwritten. This zipped results file will contain results and a snapshot of the Test Suite/Rule Application at the time of test execution.
/l RuleApplicationLocationURI	If the Rule Application is saved in irCatalog: • Catalog Service Address, including "catalog" prefix +

	Rule Application Name (catalog.http://Server/InRuleCatalog/Service.svc/ExampleRuleApp) • Catalog Username/Password (username/password) If the Rule Application is saved to file: • File system path (C:\RuleApps\ExampleRuleApp.ruleappx)

Execute a **couple of test cases** from a Test Suite.

```
> irVerify /s Example.testsuite /n ExampleTestCase1,ExampleTestCase2 /l ExampleRuleApp.ruleappx
```

Execute **all test cases** from a Test Suite and **save the results**.

```
> irVerify /s Example.testsuite /r Example.testresults /l ExampleRuleApp.ruleappx
```

Execute **all test cases found in the folders** specified and their subfolders.

```
> irVerify /s Example.testsuite /f ValidationTestFolder,PromotionTestFolder /l ExampleRuleApp.ruleappx
```

Execute test cases for a **Rule Application that is stored in irCatalog**.

```
> irVerify /s Example.testsuite /l catalog://username:password@Server/InRuleCatalog/Service.svc/ExampleRuleApp
```

Identical to the command above but uses the explicit form of the scheme to **indicate it is using HTTP** as opposed to HTTPS.

```
> irVerify /s Example.testsuite /l catalog.http://username:password@Server/InRuleCatalog/Service.svc/ExampleRuleApp
```

Execute test cases for a Rule Application that is stored in **irCatalog that is hosted on a HTTPS host**.

```
> irVerify /s Example.testsuite /l catalog.https://username:password@Server/InRuleCatalog/Service.svc/ExampleRuleApp
```

Execute test cases for a Rule Application that is stored in irCatalog that uses **single-sign-on (SSO)**.

```
> irVerify /s Example.testsuite /l catalog.http://Server/InRuleCatalog/Service.svc/ExampleRuleApp
```

Execute test cases from a **Test Suite that was created via the SDK**. Using the SDK, you can save the Rule Application location within the Test Suite file. The Rule Application must exist on the file system relative to where the Test Suite file was originally saved.

```
> irVerify /s Example.testsuite
```

For simplicity, all the examples above assumed that everything is saved within one folder. If this is not the case, you may use the full file path or the relative file path for the Test Suite, test results, and Rule Application. In the following example, the command is being executed from C:\InRule where irVerify.exe is located. Test Suite uses the full file path and Rule Application uses the relative file path.

```
>irVerify /s C:\RuleApps\Example.testsuite /l ..\RuleApps\ExampleRuleApp.ruleappx
```

Part IX

Rule Tracing

9 Rule Tracing

Overview

The InRule Rule Engine provides the following ways to obtain detailed logs of events that occur during execution to aid the debugging of a rule application:

- The RuleExecutionLog class provides information on state changes and rules firing.
- Rule tracing provides the chronological sequence of activities, including the constituent components the rule engine uses to evaluate a rule

As with the RuleExecutionLog, rule tracing is accessible after either an ApplyRules or ExecuteRuleSet has been issued to the RuleSession. By default, rule tracing is disabled; it can be enabled either through irVerify or programmatically through irSDK. Unless the trace is persisted, it will be replaced during a subsequent rule execution. Rule tracing does not have to capture every activity that occurs during rule execution; it can be selective based on a filter constructed prior to rule execution. The output of the rule trace is presented in XML form. A user may elect to output the rule trace as XML in its entirety, or navigate to and output a specific section using a trace reader provided by irSDK.

The following topics provide assistance for working with rule tracing:

- Rule Tracing Concepts
- Rule Tracing SDK Overview
- Working with the Trace Viewer through irSDK

A developer can also work with irSDK to create and persist rule traces using a variety of configuration options. Refer to the Runtime API Examples for further detail.

Long-running / Complex Rule Executions

Rule Tracing has been optimized to minimize memory use and reduce performance overhead. However, there is always going to be some noticeable drag on performance when tracing is enabled. By default, tracing is disabled (TraceFrameType.None) so tracing is an explicit opt-in requirement.

A rule execution with all frame types enabled, producing several hundred thousand or even million trace frames may be unmanageable when used within irVerify. Using the irSDK to write XML to a file instead of an in-memory object or string is sometimes a better approach.

The total number of frames reported by the RuleExecutionLog.TotalTraceFrames property provides an idea of the trace size before attempting to render it. In some cases a large trace can take several minutes to render even on a fast machine. The value of TotalTraceFrames can be checked before calling RuleExecutionLog.GetExecutionTrace().

Temporary Trace File Location

During rule execution, the trace files will be persisted under the InRule temporary file location. By default this located at %TEMP%\InRule\Temp.

9.1 Rule Tracing Concepts

Trace Frame

Activities logged by the rule tracing are represented as Trace Frames. A frame may encapsulate aspects and other frames.

A frame has the following mandatory attributes:

- FrameId - The unique frame identifier.
- Timestamp - The point in time the activity was recorded.
- FrameType - The type of activity being traced. (E.g. SetValue, EvaluateRule,

ValueChanged.)

It may also have the following optional attributes:

- SourceDefId - The GUID of the Def in the RuleApplication relating to this activity.
- StateId - The ElementIdentifier relating to this activity. (E.g. {Entity1::1}/Entity1/Calc1)

Aspect

Aspects represent properties specific to each rule engine activity. They exist as a name-value pair.
E.g. Name=Result, Value=1234.

Frame Type

Each frame has a type to describe the type of activity being traced. The frame types to be recorded can be configured using irSDK at either a high-level (RuleExecutionSettings.EnableRuleTracing) for trace event visualization, or fine-grained (RuleExecutionSettings.TraceFrameTypes) for low-level debugging. irVerify uses the high-level settings.

Trace Events

If the trace is viewed from irVerify (or the EventReader is used from irSDK), the Trace Frames are converted into a high-level list of Trace Events. A Trace Event is the result of a collection of Trace Frames. The events closely represent the logical flow of the authored rules and can be viewed in irAuthor to assist debugging rule execution.

A Trace Event may encapsulate a list of Attributes and Evaluations.

- Attributes - A name-value pair belonging to the event.
- Evaluations - The decomposition of source syntax rules and their evaluated results.

A Trace Event has the following attributes:

- EventId - The unique event identifier.
- ParentEventId - The parent event identifier.
- Timestamp - The point in time the event’s root Trace Frame was recorded.
- EventFraneId - The identifier of the event’s root Trace Frame.
- DisplayName - The display text of the original authored rule.
- Result - The result of the rule or action, if there is one.
- Context - The context under which the event took place. (E.g. Entity1, RuleSet1)
- StateId - The ElementIdentifier relating to this event. (E.g. {Entity1::1}/Entity1/Calc1)

Part X

Attribution

10 Attribution

InRule, InRule Technology, irAuthor, irVerify, irServer, irCatalog, irSDK and irX are registered trademarks of InRule Technology, Inc. irDistribution and irWord are trademarks of InRule Technology, Inc. All other trademarks and trade names mentioned herein may be the trademarks of their respective owners and are hereby acknowledged.

Index

inrule

For more information about InRule, please visit https://www.inrule.com.

For support please visit https://support.inrule.com

If you would like to report an issue or request a feature enhancement, please email support@inrule.com.

Date of Publication: 3/22/2021