ForceRuler

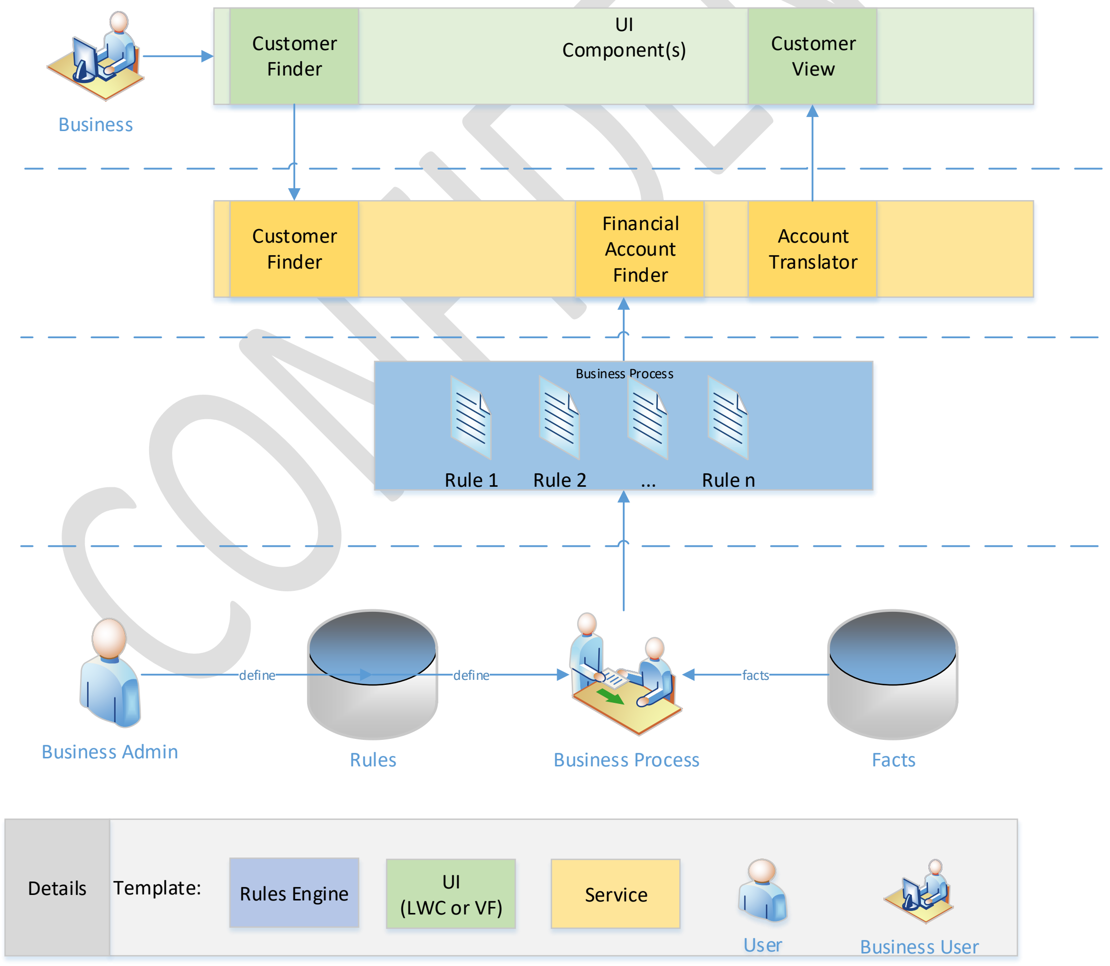
ForceRuler allows customers to keep their Apex components but curbs the use. How? Following a DROOLS like structure (*Rule* … *When* … *Then* … *End*), rules can be created to perform Business Functions. With a proper User Interface (non-Custom Metadata), Business Users can take advantage of the underlying Apex Assets (should they wish) while maintaining a high-level of functionality.

Rules can be created to perform both evaluation and processing. The rules can be used to filter Facts (i.e. SObject or Generalized data) as well as change facts and/or perform internal or external functions.

The Best of Both Worlds: The Developers and the Business Users

Business Users and Developers clash in their understanding when it comes to applying Business Functionality. To help mitigate this, *and stay on the platform*, allow Business Rules to be crafted and processed in Salesforce. Yes, you have Flow, Process Builders, etc. but *you cannot dynamically craft a flow, process builder or Apex at runtime*.

With *ForceRuler* we can! In fact, we can craft, parse and provide (from cache) in a performant way.



Craft Business Rules,

```
rule "Suggest Manager Role"
when
            Context(sync=false, debug=true);

            Contact(experienceInYears > 10 && email.isBlank() == false );
            Contact(currentSalary > 100000 && currentSalary <=  250000);
            Contact(role.contains("manager") == true);
then
            fr.save("Manager");
            Contact(hasSuggested=true);
end;
```

Now, execute; reducing the Apex footprint for companies, providing *Click, No Code*, in a controlled manner. For example, in Apex, a small amount of code can run (which I have not ***yet*** created a LWC wrapper for this):

```
eval_Facts facts = new eval_Facts(Trigger.New);
sf_IRuleEngineWorker worker = sf_RuleWorkerFactory.create('Suggest Manager Role',facts);
// run invokes both the WHEN and THEN section
eval_Status status = worker.run(); // or could just evaluate, worker.evaluate(); [WHEN Section]
```

Figure 1 Apex Code

Performance

ForceRuler provides the ability to craft rules at runtime or pull from Custom Metadata. In addition, it attempts to utilize Platform Cache, as well as transaction caching, to avoid re-creating a parse tree. Caching provides a means to improve performance. Please know, caching can evict at any time and has thresholds that may cause the behavior to change.

The overall goal was to allow these rules to work with Bulk in mind. However, depending on the complexity of the rule this may not true. Measuring performance on a cached rule took on average approximately <300 milliseconds. An unparsed (non-cache) rule performance will vary between 400 – 1100 milliseconds and depends on the number of expressions in the rule.

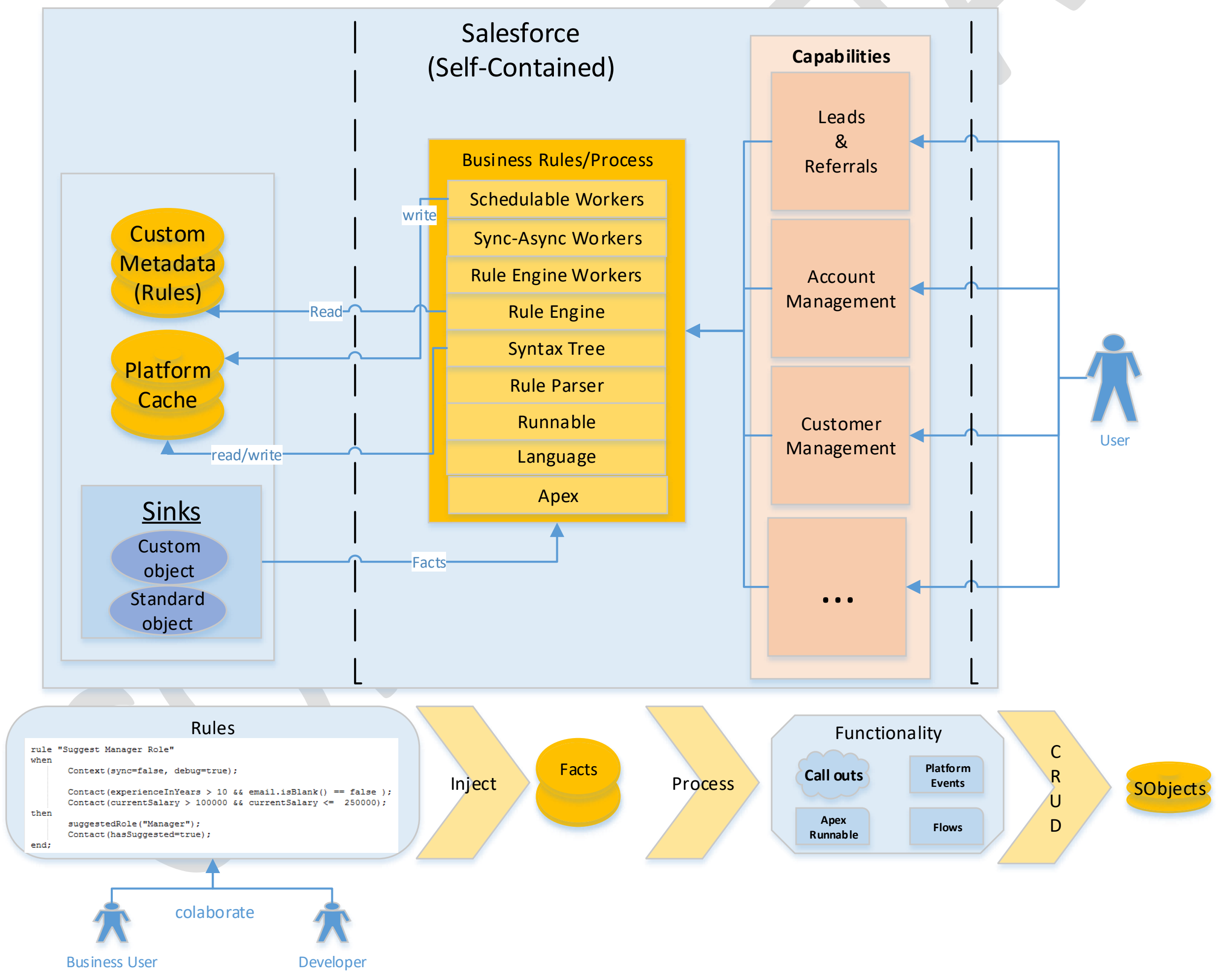
Advantages

ForceRuler provides the following advantages,

- Create rules on the fly, allowing external and internal callers to craft business rules (as needed)
- Performant (underlying Apex and uses Platform Cache)
- Rules can be created by Technical and non-Technical personnel
- Allow Facts to be evaluated/filtered (both statically and dynamically) [WHEN Section]

- Allow Facts to be evaluated and process (both statically and dynamically) [WHEN & THEN Section]
- Intuitive (simple to use)
- Reduces Apex Footprint
- Allows work to be run Asynchronously or Synchronously
- Extensible (allowing Apex classes to be injected)
- Provides additional classes[1] for enhanced functionality:
    - Publish Events
    - Big Object Audit
    - Etc.

High-Level Technical Breakout



```
rule "Suggest Manager Role"
when
        Context(sync=false, debug=true);

        Contact(experienceInYears > 10 && email.isBlank() == false );
        Contact(currentSalary > 100000 && currentSalary <=  250000);
then
        suggestedRole("Manager");
        Contact(hasSuggested=true);
end;
```

[1] This functionality has yet to be completed

What about Flows[2]?

ForceRuler can invoke Flows, as needed. This does not negate the client assets. What ForceRuler provides is a pseudo-Business-Developer aspect. Rules can be easily changed to various needs. The language is simple and intuitive. Adding a presentation layer to craft rules will make the process even more intuitive.

Used in a Trigger

Below is a sample trigger for Contact (handling Before Insert/Update).

```
/**
 * Note           : This is Just a Sample! We would normally use a Trigger Framework to call into!
 * @description: Sample Rule Trigger to process Before Insert/Update. Small Apex footprint
 * @author      : Bill Anderson
 * @group       : Sample
 **/
trigger Contacts on Contact (before insert, before update) {
   // our contact rule name
   final string RULE_NAME = util_Constants.CONTACT_RULE_NAME;

   if (Trigger.isInsert) {
      if (Trigger.isBefore) {
         eval_status status = (new sf_RuleTriggerRunner(RULE_NAME,Trigger.New)).run();
         system.debug('+++++ [Bef Ins] FACTS     :' + status.evaluatedSuccessfulFacts);
         system.debug('+++++ [Bef Ins] PROC FACTS:' + status.processedFacts);
         system.debug('+++++ [Bef Ins] SUCCESS   :' + status.success());
      } // end of before update
   } else if ( Trigger.isUpdate ) {
      if (Trigger.isBefore) {
         // Process before update
         eval_status status = (new sf_RuleTriggerRunner(RULE_NAME,Trigger.New)).run();
         system.debug('+++++ [Bef Upd] FACTS     :' + status.evaluatedSuccessfulFacts);
         system.debug('+++++ [Bef Upd] PROC FACTS:' + status.processedFacts);
         system.debug('+++++ [Bef Upd] SUCCESS   :' + status.success());
      }
   } // end of update

}// end of trigger
```

[2] Have not implemented yet; Work in Progress

Used in Change Data Capture

Often there is a need to evaluate changes in Salesforce and Change Data Capture can be used along with rules to perform that task.

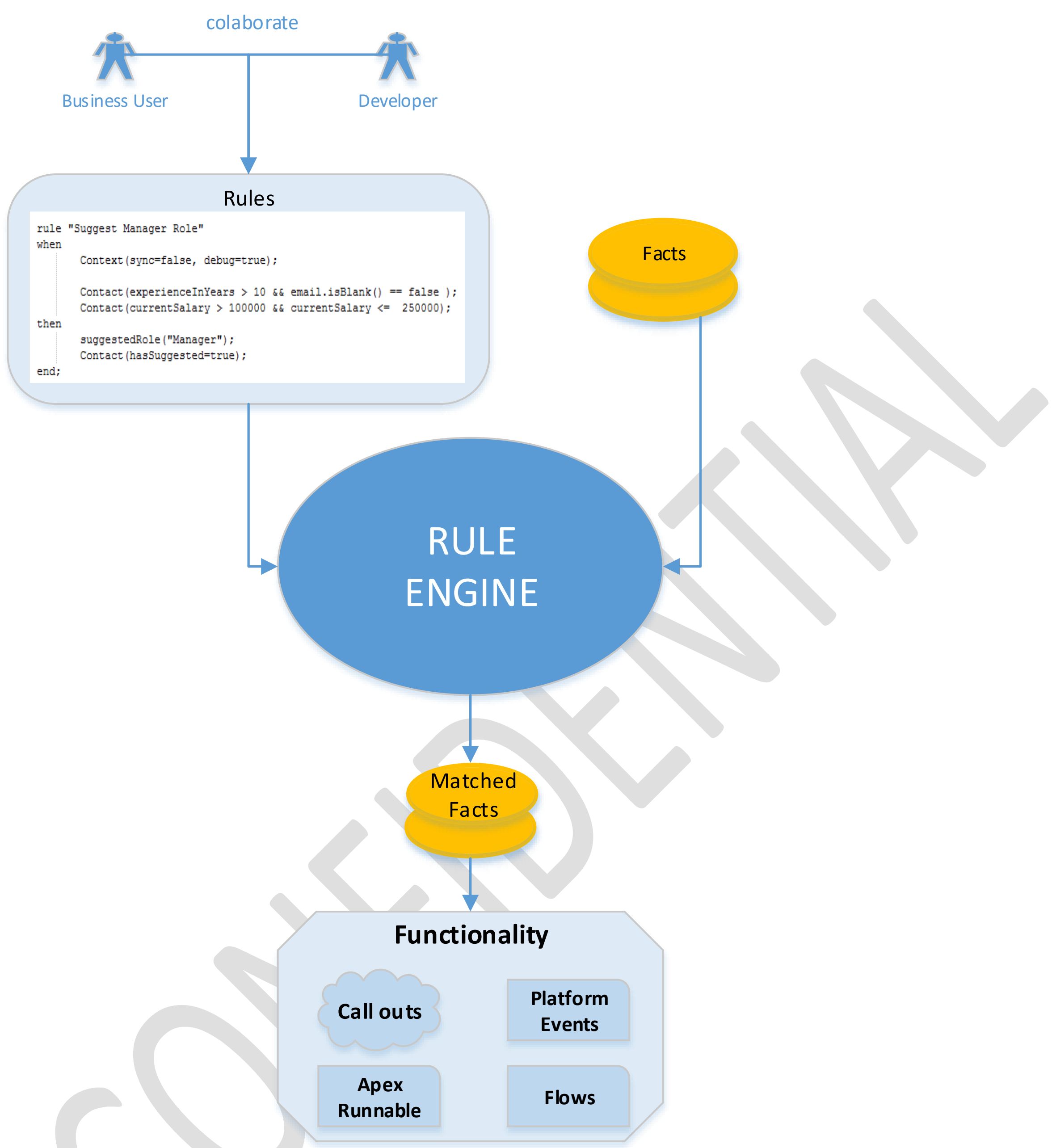
```
trigger AccountChangeEventTrigger on AccountChangeEvent (after insert) {

    String PE_Rule ='rule "AccountChangeEvent Rule"  ' +
                     '     Context(sync=true, debug=true); ' +
                     ' when ' +
                     '      AccountChangeEvent("update" == changetype && true == changedfie
lds.contains("industry") &&  industry!="banking" ); ' +

                     ' then  ' +
                     '      fr.out(); ' +
                     ' end';
    // this processes the CDC rule ...
    eval_status status = (new sfr_RuleTriggerRunner(PE_Rule,Trigger.New,false)).run();
    system.debug('+++++ [CDC] FACTS COUNT  :' + status.evaluatedSuccessfulFacts);
    system.debug('+++++ [CDC] PROC FACTS   :' + status.processedFacts);
    system.debug('+++++ [CDC] SUCCESS      :' + status.success());
    system.debug('+++++ [CDC] FACTS        :' + status.theFacts);
    system.debug('+++++ [CDC] STATUS       :' + status);
}
```

General Attributes

- Rules → Flows
- Rules → Runnable (Apex)
- Rules → Asynchronous / Synchronous
- Rules → SObject (CRUD)
- Rules → CDC



```
rule "Suggest Manager Role"
when
        Context(sync=false, debug=true);

        Contact(experienceInYears > 10 && email.isBlank() == false );
        Contact(currentSalary > 100000 && currentSalary <=  250000);
then
        suggestedRole("Manager");
        Contact(hasSuggested=true);
end;
```

SMART Monitoring

With the ability to create rules dynamically, you can generate rules as needed. A proactive system without the need to react after the event. Companies would like their Sales or Service to change based on incoming data/facts. This ability is not possible unless you can dynamically inject rules/functionality at runtime; then it may be too late. ***ForceRuler*** allows rules to be created as needed.

For example, you could monitor your Sales or Service data via Change Data Capture (CDC). Based on certain data changes, rules could be generated to take the appropriate action; i.e.

- Sales Volume,

- Sale to a certain vendor
- Sales or Service Threshold,
- Etc.

Rule Chaining

Often Business Functionality changes and there is no mechanism to adjust whether Apex, Flow, Process Builder, etc. ***ForceRuler*** provides the ability to chain rules. For example, a Business may need to adjust discounts on various Sales promotions based on Volume. As volumes change, the Business wants to react appropriately. The initial Business Requirements defining the ***anticipated*** *volumes* may have been hard coded in Apex or Flow. But, using SMART Monitoring, a Business rule could be adjusted to adopt to your Sale Volumes and either replace a current rule or chain onto the current rule.

- *Rule 1 → Rule 2 →… → Rule N*

Rules and Change Data Capture [CDC]

Another example of CDC, ForceRuler can also filter and process CDC triggers. For example, given the following rule,

```
/**
 * Note              : This is Just a Sample! We would normally use a Trigger Framework to call into!
 * @description: Sample Rule Trigger to process changes from a CDC Event
 * @author          : Bill Anderson
 * @group           : Sample
 **/
trigger AccountChangeEventTrigger on AccountChangeEvent (after insert) {
    //
    // This rule states WHEN an Account is updated and is an AccountChangeEvent
    // (As this may be a channel) AND the Industry field CHANGED, THEN
    // Call forcerule function to output data.
    //
     final String    PE_RULE ='rule " AccountChangeEvent Rule"  ' +
                          '    Context(sync=true, debug=true); ' +
                          ' when ' +
                          '    AccountChangeEvent ("update" == changetype && true ==
changedfields.contains("industry") ); ' +
                          ' then  ' +
                          '        fr.out("AccountChangeEvent")  ' +
                          ' end';
   //
  // Note, the invocation and processing are the same as any other rule. The only distinctions in the
  // CDC event above is the use of keywords (changetype and changedfields)
  //
   eval_status status = (new sf_RuleTriggerRunner(PE_RULE,Trigger.New,false)).run();
   system.debug('+++++ [CDC] FACTS     :' + status.evaluatedSuccessfulFacts);
   system.debug('+++++ [CDC] PROC FACTS:' + status.processedFacts);
   system.debug('+++++ [CDC] SUCCESS   :' + status.success());

}// end of trigger
```

CDC events require a specialization. In the above sample, the domain is *Change Data Capture Event*. This allows the parser to treat the components within the parenthesis as CDC Events. The other three special values are:

- **changetype** – indicates if this is a **CREATE**, **UPDATE**, **DELETE**, **UNDELETE**, **GAP_CREATE**, etc.
- **objectName** – indicates the type of event
- **changedfields.contains** – indicates whether a field (i.e. *Industry*) has changed.

BNF Syntax and Examples

Below is the BNF Syntax along with examples.

```
<rule-syntax>  ::= rule <rule> context <context-items> when <when-items> then <then-items> end ";"
<rule> ::= '"' <rule-name> '"' <whitespace>
<global-items> ::= "(" <opt-whitespace> <global-list> <opt-whitespace> ")" ";"
<global-list>  ::= <globla-term> | <opt-whitespace>
<global-term> <globla-name> <assignment> <function> | <text1> | <digit>
<global-name>
                "$"<character>
<context-items> ::= "(" <opt-whitespace> <context-list> <opt-whitespace> ")" ";"
<context-list>  ::= <context-term> | <opt-whitespace>
<context-term> <context-name> <assignment> <boolean> | “queue”
<context-name>
                "debug"
                "sync"
               “ProcessType”
<when-items> ::= <domain-name> "(" <opt-whitespace> <when-expression> <opt-whitespace> ")" ";"
<when-expression> ::= <when-list> | <when-list> "||" <when-list> | <when-list> "&&" <when-list>
<when-list> ::= <term> <operator> <term> |  <when-list> <operator> <term>
<term>         ::= <literal> | <date> | <boolean> | <when-term>
<when-term>  ::=
                <variable>
                date.today()
                date.tomorrow()
                date.yesterday()
                date.this_year()
                date.last_year()
                isblank()
                contains()
<then-items> ::= <domain-name> "(" <opt-whitespace> <then-expression> <opt-whitespace> ")" ";"
<then-expression> ::= <then-list> | <then-list> "||" <then-list> | <then-list> "&&" <then-list>
<then-list> ::= <variable> <assignment> <literal> |  <function>
<function>  ::=  <function-name> "(" <function-arguments> ")" ";"
<function-name> ::= <letter> | <letter> <function-name>
<function-arguments> ::= '"' <text2>  '"' | <function-arguments> ","
<variable>  := <domain-name> "." <field-name> | <field-name>
<boolean>  ::= true | false
<assignment>  :: = "="
<operator>                ::= ">" | "<" | "==" | ">=" | "<="
<literal>       ::= '"' <text1> '"' | "'" <text2> "'"
 <text1>        ::= "" | <character1> <text1>
 <text2>        ::= '' | <character2> <text2>
<date>      ::= <month> "/" <day> "/" <year>
<month>  ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12
<day>  ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 ... | 31
<year> ::= <digit> <digit> <digit> <digit>
<domain-name>
                Account
                Contact
                Opportunity
                Case
                 :
                Any-Salesforce-Standard-Or-Custom-Object-Name
<character>     ::= <letter> | <digit> | <symbol>
<whitespace> ::= " " | "\n" | "\t"
<opt-whitespace> ::= " " <opt-whitespace> | ""
<letter>       ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" |
"X" | "Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" |
"y" | "z"
<digit>        ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<symbol>        ::=   " " | "!" | "#" | "$" | "%" | "&" | "+" | "," | "-" | "." | "/" | ":" | ";" | "_" |
<character1>    ::= <character> | "'"
<rule-name>     ::= <letter> | <rule-name> <rule-char>
<rule-char>     ::= <letter> | <digit> | "-"
<text1>        ::= "" | <character1> <text1>
```

Examples

Below are various sample rules and expressions.

Example Rules

Example rule for Contact SObject (along with custom fields):

```
rule "Suggest Manager Role"
when
        Contact(experienceInYears__c > 10 || experienceInYears__c < 1);
        Contact(currentSalary__c > 1000000 && currentSalary__c <=  2500000 || year__c == 2020);
then
        Contact(currentSalary__c =10, year__c=200, experienceInYears__c ="2010");
end;
```

Example rule for Contact SObject, ensuring any last name equal to *Chopra* and fist name equal to *wes* will have its email set to roadrunner@acme.com.

```
rule "Contact Rule Insure Email"
when
  Contact( lastname == "Chopra" && firstname == "wes" );
then
  Contact( email = "roadrunner@acme.com");
end;
```

Example rule for Contact SObject, ensuring various options are true will have its email set to coyote@acme.com.

```
rule "Contact Contains Rule"
when
        Contact( Birthdate < Date.today() && Birthdate > 12/12/1900 );
        Contact( lastname.isBlank() == false && firstname.contains("boss") == true );
then
        Contact(email = "coyote@acme.com");
end
```

Example rule for Account SObject, ensuring if the name is "*testing*" change to "*test change*". And, if it is not in a Trigger, save the information. Note, ***fr.save()***, is an alias function. Functions are aliases that are mapped to special functionalities. See Special Functions section for more information.

```
rule "Account Check Name is Testing"
   Context(sync=false, debug=true, ProcessType=queue);
when
        Account(name.contains("testing") == true );
then
        Account(name = "test change");
        FR.save();
end
```

Example rule for Account SObject, ensuring if the name is "*flow*" call a flow. Note, ***flow.run(…)***, is an alias function map to invoke a Flow. Some functions are alias; See Special Functions section for more information.

```
rule "Account Check Name is Testing"
        Context(sync=true, debug=false);
when
        Account(name=="flow"  );
then
        flow.run("","THE_FLOW_API_Name", "arg1:somedata", "arg2:moredata");
end
```

Example rule for Account SObject using Global section to define a value for use within the rule.

```
rule "Account Check Name is Testing"
        Global( $name = "flow");
        Context(sync=true, debug=false);
when
        Account(name == $name);
then
        flow.run("","THE_FLOW_API_Name", "arg1:somedata", "arg2:moredata");
end
```

Example Expressions

Expression	Comment
Contact(birthday < "1/21/1990");	Birthday field in Contact less than a date
Contact(birthday < date.today());	Birthday field in Contact less than today
Contact(birthday >= date.yesterday());	Birthday field in Contact greater than or equal to yesterday
Contact(birthday > date.last_year());	Birthday field year in Contact greater than last year
Contact(lastname.contains("anderson") == true);	Lastname field in Contact contains *anderson*
Contact(endatetime > "11/06/1999, 01:00 PM");	EndDateTime greater than the date time provided

myCallableClass("arg1", "arg2");	Make a call to the Apex Class, ***myCallableClass***, passing in two arguments

Special Functions

ForceRuler allows functionality to be alias to Apex classes. This allows the rule function to remain the same yet the underlying implementation can be changed internally. There are three built-in functions[3], but one has the ability (via custom metadata) to change the underlying class called.

Alias	**Class Name**	**Description / Comment**
fr.save()	*sf_RunnableSave*	Apex Class which supports the *sf_RERunnable*[4] Interface. Will save the data to Salesforce
fr.update()	*sf_RunnableSave*	Apex Class which supports the *sf_ RERunnable* Interface. Will udpate the data to Salesforce
flow.run()	*sf_RunnableFlowCaller*	Apex Class which supports the *sf_ RERunnable* Interface. Will invoke a specific flow, passing in user-defined arguments and Facts

User Interface

ForceRuler is an application to view, create, edit, test and run rules. There are three modules found in the *ForceRuler* Application.

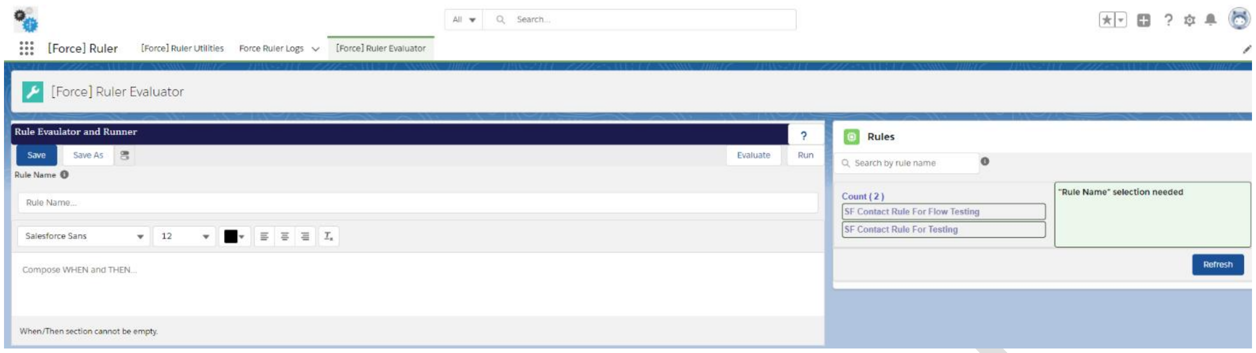
1) Create, Run and Evaluate Rule Module
2) Rule Viewer Module
3) Rule Log Viewer

These User Interfaces were developed to ease the burden of creating, editing, testing and running rules. The generic interface found Custom Metadata was not appropriate for the creating, editing, testing and running rules. As such, several modules were created to ease the burden; and, can still be considered a work in-progress (WIP).

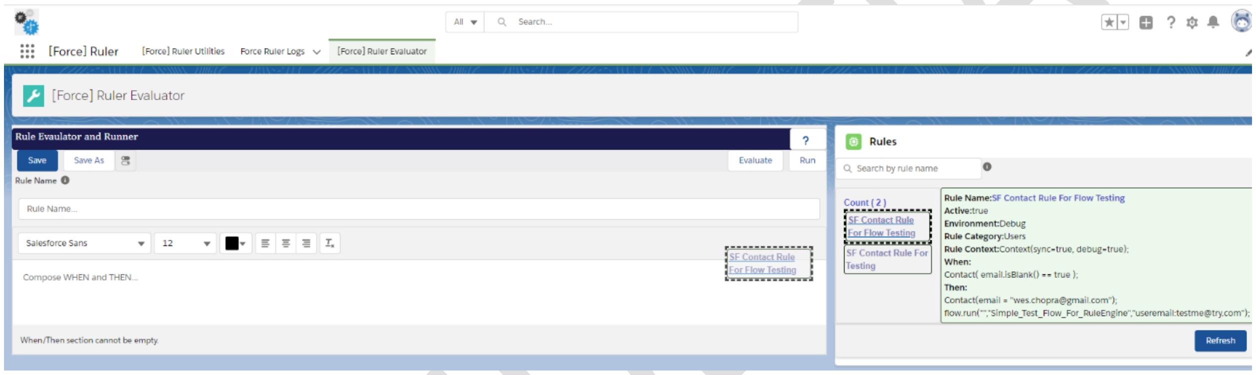
Create, Run and Evaluate Rule Module

This module allows you to create, view, edit new and existing Rules.

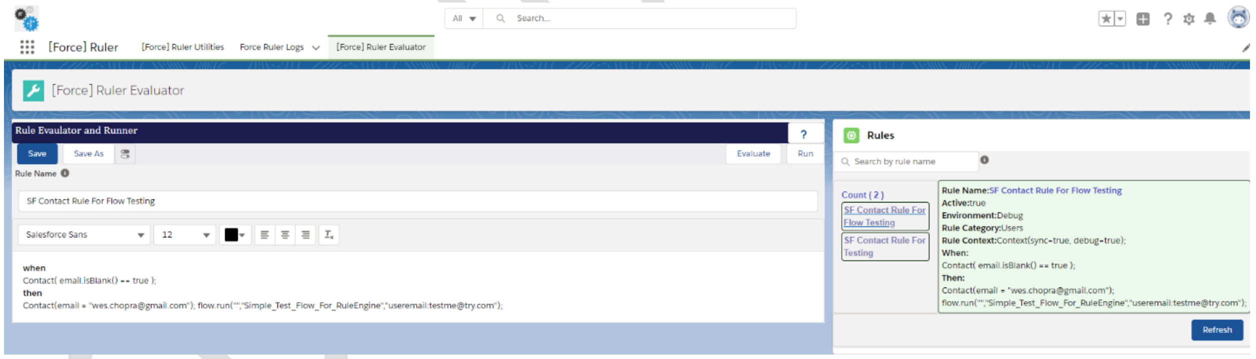
[3] Function or class names are **NOT** case-sensitive.
[4] All Invocable Apex classes MUST inherit from *sf_IRERunnable*.



Drag and Drop Current Rules for Edit and Evaluation



Rule Name: SF Contact Rule For Flow Testing
Active: true
Environment: Debug
Rule Category: Users
Rule Context: Context(sync=true, debug=true);
When:
Contact(email.isBlank() == true);
Then:
Contact(email = "wes.chopra@gmail.com");
flow.run("","Simple_Test_Flow_For_RuleEngine","useremail:testme@try.com");

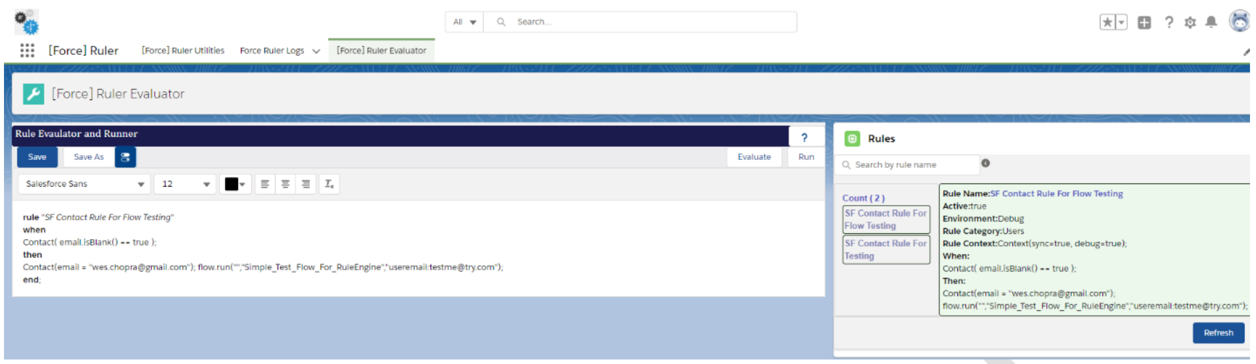


Rule Name: SF Contact Rule For Flow Testing

```
when
Contact( email.isBlank() == true );
then
Contact(email = "wes.chopra@gmail.com"); flow.run("","Simple_Test_Flow_For_RuleEngine","useremail:testme@try.com");
```

Rule Name: SF Contact Rule For Flow Testing
Active: true
Environment: Debug
Rule Category: Users
Rule Context: Context(sync=true, debug=true);
When:
Contact(email.isBlank() == true);
Then:
Contact(email = "wes.chopra@gmail.com");
flow.run("","Simple_Test_Flow_For_RuleEngine","useremail:testme@try.com");

The screen view below shows the Rule, **SF Contact Rule For Flow Testing,** dragged and dropped into the *Rule Evaluator and Run* module. This view is the *Free-Form* view. A simple editor to allow for those most familiar with the syntax and/or quick edits.



```
rule "SF Contact Rule For Flow Testing"
when
Contact( email.isBlank() == true );
then
Contact(email = "wes.chopra@gmail.com"); flow.run("","Simple_Test_Flow_For_RuleEngine","useremail:testme@try.com");
end;
```

```
Rule Name:SF Contact Rule For Flow Testing
Active:true
Environment:Debug
Rule Category:Users
Rule Context:Context(sync=true, debug=true);
When:
Contact( email.isBlank() == true );
Then:
Contact(email = "wes.chopra@gmail.com");
flow.run("","Simple_Test_Flow_For_RuleEngine","useremail:testme@try.com");
```

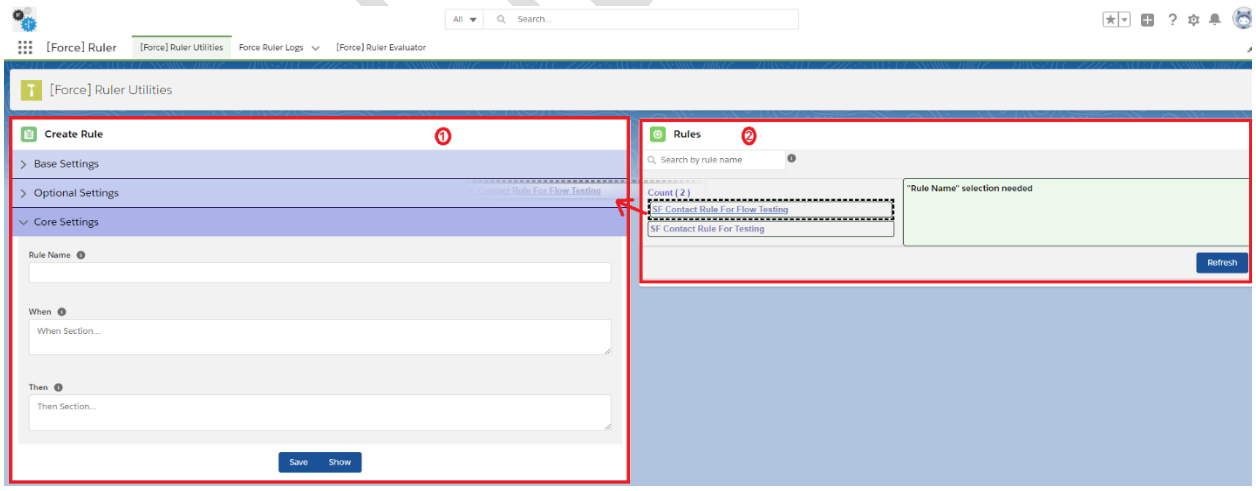
Create Rules and Rule View

This module provides a more guided approach for those not yet familiar with the rule format. The *Create Rule* module provides a guided approach to creating a rule. While the Rules module provides a list view of current Rules. The current Rules can be defined in three different environments (1) ***Test***, (2) ***Debug***, (3) ***Production***. Each environment is distinguished by the following:

- **Test –** Visible while Unit Testing
- **Debug –** Visible in a Scratch or Sandbox Org.
- **Production –** Visible in a Production Org.

Create/Edit and View List of rules provides a simple user interface to let you view and drag-n-drop rules into an edit window. The example below shows two modules. The first module (1), allows you to create and edit rules. The second module (2), allows you to view all current rules defined for that environment.

Selecting a rule in rules module (2) and dragging the contents to the create/edit module (1) provides a simple means to edit existing rules.


[Force] Ruler
[Force] Ruler Utilities
Force Ruler Logs
[Force] Ruler Evaluator
Create Rule
1
Base Settings
Optional Settings
Core Settings
Rule Name
When
When Section...
Then
Then Section...
Save
Show
Rules
2
Search by rule name
Count (2)
SF Contact Rule For Flow Testing
SF Contact Rule For Testing
"Rule Name" selection needed
Refresh