Constraints

Overview

In specific scenarios, system allows user to define what parameter values can never be tested together or in the other hand, scenarios where parameter values can only be tested together.
For that, user can apply constraints to support these cases.

So that the model can understand the requirements, you have the option to define invalid constraints (values that can never be tested together), and/or bound constraints (values that can only be tested together) to train the model.

Invalid constraints

Invalid constraints restrict parameter values that can **never** be tested together.

In this example, let's assume Internet Explorer (IE) is not supported on Apple computers so it would be impossible for a tester to execute a test case that instructed them to launch IE from a Mac running on its native operating system. Accordingly, we do not want any tests generated to include combinations such as "OS X" and "IE8" in the same test case.

On the Rules ->Constraints screen, you will need to click on two red X's. Hover over the first Value of the Invalid Pair and click on the red X that appears

Find the second Value that can never appear together with the first one, hover over it, and click on the red X to create your "Invalid Pair"

As you enter your Constraints, you will see them listed to the left:

After you enter these two Invalid Pairs, clicking on the "Scenarios" button will create a completely new set of test cases that excludes those two pairs of Values (and only those two pairs of Values). Each test with 'OS X' as the operating system will have a browser other than IE.
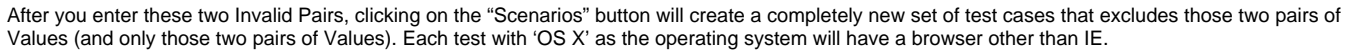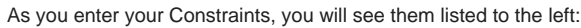
As you use the Invalid Pair feature and the related Bound Pair one, keep in mind these usage tips:

Do you need to add *a lot* of Invalid Constraints?

- Imagine you had an application that asked, "Would you like to add a hotel to your travel reservation?" Any scenarios with "Do Not Add a hotel" would require many, many different Invalid Constraints between things like "Do Not Add a Hotel" and "Hotel Chain Preference = X". For situations like those, the Bound Constraint feature will help you accomplish your constraint handling much faster.
- If you have more than 10 or so Invalid Constraints (or Bound Constraints) in your plan, you might find that it is faster to export your plan into Excel and document your paired values in Excel. If so, make sure to both (1) add multiple paired values in Xray Test Case Designer before you export into Excel (so you can copy and paste the Xray Test Case Designer formatting for Invalid Constraints and /or Bound Constraints), and (2) ensure that you use the exact spelling of Values (e.g., 'cutting and pasting' Values will usually be safer than typing)

Watch out for "Not Applicable" Values.

- Do you need to add some of them to your plan? Do you need to invalidate the combination of "Not Applicable" with other specific Values?
- Especially watch out for situations where you have multiple related "Not Applicable" values in a plan. Would it make sense to create a "Bound Constraint" between 'Hotel Chain Preference = Not Applicable' and 'Type of Room' = Not Applicable'?

Bound constraints

Bound constraints restrict parameter values that can only be tested together.

Consider these parameters & values:

Parameters <

Customer Type (2)	Regular	VIP		
Type of Flight (2)	Domestic	International		
Nights at Destination (3)	1	2-7	8-14	
Add a hotel (2)	Add a hotel	Do not add a hotel		
Hotel Chain Preference (4)	Marriot	Hilton	Motel One	Vivanta by Taj
Type of Room (3)	Least Expensive	Moderately Expensive	Most Expensive	

Sidebar: Parameters, RULES, SCENARIOS, SCRIPTS, ANALYSIS, REVIEW, Share

You will have the following issue when you click on the "Scenarios" button:

#	Customer Type	Type of Flight	Nights at Destination	Add a hotel	Hotel Chain Preference	Type of Room
2	VIP	International	2	Do not add a hotel	Hilton	Least Expensive
5	Regular	Domestic	12	Do not add a hotel	Marriot	Most Expensive
6	VIP	Domestic	1	Do not add a hotel	Vivanta by Taj	Least Expensive
8	VIP	Domestic	1	Do not add a hotel	Motel One	Moderately Expensive
12	VIP	Domestic	2	Do not add a hotel	Vivanta by Taj	Most Expensive
13	VIP	Domestic	14	Do not add a hotel	Hilton	Moderately Expensive

Even though the test calls for not adding the hotel, the data row proceeds to specify the preference and type of room, which is clearly incorrect.

To solve this, you first need to add "Not Applicable" as parameter values

Customer Type (2)	Regular	VIP		
Type of Flight (2)	Domestic	International		
Nights at Destination (3)	1	2-7	8-14	
Add a hotel (2)	Do not add a hotel	Add a hotel		
Hotel Chain Preference (5)	Not Applicable	Marriot	Hilton	Motel One	Vivanta by Taj
Type of Room (4)	Not Applicable	Least Expensive	Moderately Expensive	Most Expensive	

This is because something will need to appear in test cases that include the value "Do Not Add a Hotel".

We will want "Not Applicable" to appear for both "Hotel Chain Preference" and "Type of Room" in every scenario that includes the value "Do Not Add a Hotel".

> ⓘ *The exact syntax of the conditional value and its position in the list don't matter, it could have been "N/A" as the last value.*

Next, you'll want to make sure that "Do Not Add a Hotel" only gets paired with the "Not Applicable" Values. You have two options under the "Constraints" tab. One is quick, the other is slow. Let's see the slow option, first:

Customer Type (2)		Regular		VIP						
Type of Flight (2)		Domestic		International						
Nights at Destination (3)		1		2-7		8-14				
Add a hotel (2)	✕ **7**	Do not add a hotel	✕ **2**	Add a hotel						
Hotel Chain Preference (5)	✕ **1**	Not Applicable	✕ **1**	Marriot	✕ **1**	Hilton	✕ **1**	Motel One	✕ **1**	Vivanta by Taj
Type of Room (4)	✕ **1**	Not Applicable	✕ **1**	Least Expensive	✕ **1**	Moderately Expensive	✕ **1**	Most Expensive		

never Add a hotel = Do not add a hotel
never Hotel Chain Prefe... = Marriot
never Add a hotel = Do not add a hotel
never Hotel Chain Prefe... = Hilton
never Add a hotel = Do not add a hotel
never Hotel Chain Prefe... = Motel One
never Add a hotel = Do not add a hotel
never Hotel Chain Prefe... = Vivanta by Taj
never Add a hotel = Do not add a hotel
never Type of Room = Least Expensive
never Add a hotel = Do not add a hotel
never Type of Room = Moderately Expe...
never Add a hotel = Do not add a hotel
never Type of Room = Most Expensive
never Add a hotel = Add a hotel
never Hotel Chain Prefe... = Not Applicable
never Add a hotel = Add a hotel
never Type of Room = Not Applicable

Implied Constraints (10) >

Namely, it's adding a lot of "Invalid Constraints" as described in the previous article in this section.

How to use Bound Constraints

1 Click on the green arrow icon to the right from the first value name (it appears on hover)

2 Click on the green arrow icon to the right from the second value name (it appears on hover)

3 Confirm how you'd like to have the bound constraint operate:

Create Constraint   ✕

Add a hotel		Hotel Chain Preference
Do not add a hotel	→	Not Applicable
Do not add a hotel	←	Not Applicable
Do not add a hotel	⇄	Not Applicable

Select how these parameter values will be bound together.

Constraints limit the parameter values that will appear together in your test cases so you can prevent impossible test scenarios from being generated.

Most of the time when you have "Not Applicable" as an option, you will use "mutually bound constraints" as in this case. Here, we want "Do Not Add a Hotel" to be bound with Type of Room as "Not Applicable" AND we want to have Type of Room of "Not Applicable" to be bound with "Do Not Add a Hotel" so we mark this one as a Mutually Bound Constraint.

When you add a Bound Pair, Xray Test Case Designer will constrain the first value chosen against all the other values in the parameter of the second value chosen. In the example above, creating a bound constraint of 'Do Not Add Hotel' and 'Hotel Chain Preference' = 'Not Applicable' means you are really invalidating these options:

- 'Do Not Add a Hotel' with 'Hotel Chain Preference = Marriott' (this combination will never appear).
- 'Do Not Add a Hotel' with 'Hotel Chain Preference = Hilton' (this combination will never appear).
- 'Do Not Add a Hotel' with 'Hotel Chain Preference = Motel One' (this combination will never appear).
- 'Do Not Add a Hotel' with 'Hotel Chain Preference = Vivanta by Taj' (this combination will never appear).

And because we created a MUTUALLY BOUND constraint, we are also invalidating these combinations:

- 'Do Add a Hotel' with 'Hotel Chain Preference = Not Applicable' (this combination will never appear).

As you use the Bound Constraint feature, keep the following tips in mind:

- You can use the Invalid Constraint Feature to accomplish anything that the Bound Constraint feature can do.
- The Invalid Constraint feature is frequently less confusing for new users . Don't hesitate to use the Invalid feature instead of the Bound Constraint feature.
- If you have more than 10 or so Bound Constraints or Invalid Constraints in a plan, you might find that it is easier and faster to document your paired values in Excel. If so, we would recommend (a) adding multiple paired values before you export into Excel, and (b) ensuring that you use the exact spelling of Values (e.g., 'cutting and pasting' is usually safer than typing)
- Especially watch out for situations where you have MULTIPLE related "Not Applicable" values in a plan. Would it make sense to create a "Bound Constraint" between, say, 'Hotel Chain Preference = Not Applicable' and 'Type of Room' = Not Applicable'? In the example above, it WOULD make sense to include that Bound Constraint. Every time 'Hotel Chain Preference' = 'Not Applicable' we would want 'Type of Room' to also be 'Not Applicable' also. Similarly, every time 'Type of Room' = Not Applicable, we would want 'Hotel Chain Preference' to also be 'Not Applicable'. This is an example of a type of constraint that the human brain would handle effortlessly without even consciously applying logical rules.

"Bound Constraints" are rules for your model algorithm to ensure that two values must appear together. Knowing which type to use depends on your plan structure and business & technology requirements.

We will be using this set of values to demonstrate:

Type of Animal (3)	Cat	Dog	Horse		
Breed of Animal (5)	Siamese	Persian	Shiba Inu	Golden Retriever	Arabian

As a reminder, to apply a bound constraint, you need to hover over value 1, click the green arrow icon to the right from the value name, then do the same for value 2, and select the appropriate option from the 3 provided ones.

One-way Bound Pair

The underlying relationship type: many-to-one, represented by WHEN-THEN statements in the left column.

We would like to exclude combinations like 'Breed of Animal = Shiba Inu' and 'Type of Animal = Cat' from the generated scenarios. We have 2 breeds for each of the first 2 types of animals. Therefore, we can set up 4 one-way bound pairs from "Breed of Animal" to "Type of Animal":

Create Constraint   ✕

Breed of Animal		Type of Animal
Siamese	→	Cat
Siamese	←	Cat
Siamese	⇄	Cat

Test cases with **Breed of Animal** as **Siamese** must have **Type of Animal** as **Cat**.

You can check your logic by reading the statement at the bottom of the dialog.

The order in which you set the pair matters!

In this case, we wouldn't be able to say WHEN 'Type of Animal = Cat' THEN 'Breed of Animal = Siamese' because that would prevent 'Cat + Persian' from appearing and would leave 'Persian" without any possible pairing.

Create Constraint   ✕

Breed of Animal		Type of Animal
Siamese	→	Cat
Siamese	←	Cat
Siamese	⇄	Cat

Test cases with **Type of Animal** as **Cat** must have **Breed of Animal** as **Siamese**.

⚠ You may argue that the statement at the bottom still looks correct in this direction, however you need to keep in mind how the word "must" is interpreted by the algorithm. Since the evaluation is done at the "pair of values" level, "must have Breed of Animal as Siamese" = "must NOT have Breed as Persian, Shiba Inu, Golden Retriever, Arabian" - of which "Persian" is incorrectly excluded, therefore this direction wouldn't work for this example. We could use such direction if we had multiple types that can be with only 1 breed.

Mutually Bound Pair

The underlying relationship type: one-to-one, represented by ALWAYS-ALWAYS statements in the left column.

With Mutually Bound Constraints, the values are exclusive to each other. In this example, there is only one horse breed in the second parameter. So, with 1 constraint, we can specify that "Horse" should not be paired with "Siamese, Persian, Shiba Inu, Golden Retriever" AND "Arabian" should not be paired with "Cat, Dog" - i.e., "Horse" and "Arabian" must only be tested together.

Create Constraint ✕

Breed of Animal		Type of Animal
Arabian	→	Horse
Arabian	←	Horse
Arabian	⇄	Horse

Breed of Animal as **Arabian** and **Type of Animal** as **Horse** must only be tested together.

The final set of rules for this model (in the verbal form) would look like this:

when	**Breed of Animal**	=	Siamese
then	**Type of Animal**	=	Cat

when	**Breed of Animal**	=	Persian
then	**Type of Animal**	=	Cat

when	**Breed of Animal**	=	Shiba Inu
then	**Type of Animal**	=	Dog

when	**Breed of Animal**	=	Golden Retriever
then	**Type of Animal**	=	Dog

always	**Type of Animal**	=	Horse
always	**Breed of Animal**	=	Arabian

In essence, Bound Constraints and Invalid Constraints perform similar tasks: they ensure that certain values only or never appear together - making all generated scenarios valid within the model scope. From a different point of view, Bound Constraints and Mutually Bound Constraints actually invalidate many values and Invalid Constraints bind many values, behind the scenes.

"Skip" constraints

"Skip" constraints are only available under the Advanced Mode which is enabled per request (toggle in the top left of the Constraints screen).

"Skip" constraints allow for certain parameters to be excluded from test cases when it is not appropriate for them to appear. It is a faster alternative to the "Not applicable" + Bound Pairs approach. However, as the feature is in the beta stage, please note that there are a couple open defects, specifically around the interaction of "Skip" constraints with the regular ones. So if you encounter any issues, feel free to reach out to Support or avoid skips for the time being.

How to Use Skip Constraints

For example, let's use this plan below for a flight booking system:

Ticket Type (3)	Coach	Business	First Class
Food Choice (3)	Vegetarian	Gluten-free	No restrictions
Drink Choice (3)	Soda	Water	Coffee
Checked Bag (2)	Yes	No	
Departure City (3)	SF	NY	LA
Destination City (3)	Boston	Atlanta	Miami

In this model, only customers who are in First Class get in-flight dining. Coach and Business class customers do not get in-flight dining. Therefore, test cases with Coach and Business class customers should have no value for "Food Choice".

First, navigate to Rules Constraints. Next, click the toggle to change to Advanced Mode – that is the only way to implement "Skip" constraints:
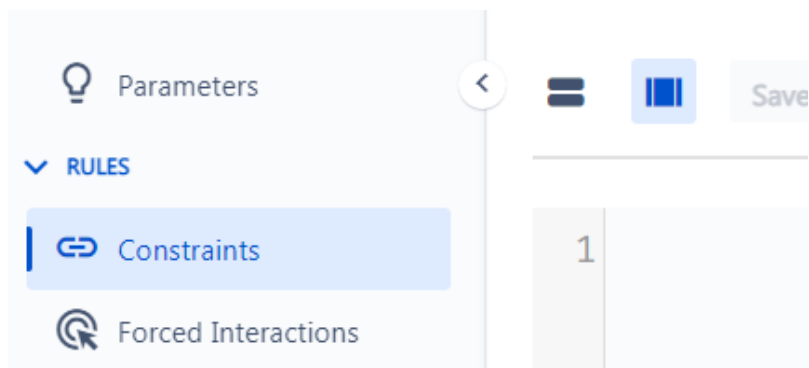
Once in Advanced Mode, we use the following syntax for "Skip" constraints:

Parameter[Parameter Value] >> Parameter(s) to be skipped

So, for our example, the syntax is this:

The ability to include multiple values in the same constraint is an aspect of the Advanced Mode that is not limited to "Skip" constraints.

Now, when we generate our test suite by navigating to Scenarios, any row that includes "Ticket Type" as "Coach" or "Business" will have no value for "Food Choice":

#	Ticket Type	Food Choice	Drink Choice
12	Business		Soda
14	Business		Water
16	Business		Coffee
11	Coach		Soda
13	Coach		Water
15	Coach		Coffee
1	First Class	Vegetarian	Soda
2	First Class	Gluten-free	Water

You can also skip multiple parameters that are listed next to each other in the Parameters screen. For example, let's say that Coach customers now get no food, drink, or checked bag. Thus, they should have blanks for each of those parameters. Since we have those 3 sequentially in the table of parameters (refer to the model screenshot at the start of this article), we can use the following syntax:

Parameter[Parameter Value] >> First Parameter to be skipped :: Last parameter to be skipped

The boundaries are inclusive, therefore, for our example, that looks like this:

```
1  Ticket Type [ Coach ] >> Food Choice :: Checked Bag
2  Ticket Type [ Business ] >> Food Choice
```

Now, when we generate our test suite by navigating to Scenarios, any rows with "Coach" for "Ticket Type" have no value for "Food Choice", "Drink Choice", or "Checked Bag":

Ticket Type ^	Food Choice ⌄	Drink Choice ⌄	Checked Bag
Business		Soda	Yes
Business		Water	No
Business		Coffee	*Yes*
Coach			
Coach			
Coach			
First Class	Vegetarian	Soda	Yes
First Class	Gluten-free	Water	No

The final component of "Skip" constraints is the 'skip to end' capability. In this case, we can tell Xray Test Case Designer that for some parameters values, skip all parameters listed later (below) on the "Parameters" page.

So, for our example, we could return to our Parameters page and reorder the elements so that "Food Choice", "Drink Choice", and "Checked Bag" are the last three parameters:

Ticket Type (3)	Coach	Business	First Class
Departure City (3)	SF	NY	LA
Destination City (3)	Boston	Atlanta	Miami
Food Choice (3)	Vegetarian	Gluten-free	No restrictions
Drink Choice (3)	Soda	Water	Coffee
Checked Bag (2)	Yes	No	

Then, we use the syntax below to skip to end:

Parameter[Parameter Value] >> First Parameter to be skipped::!!

For our example, that looks like this:
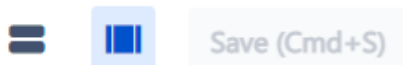

```
1  Ticket Type [ Coach ] >> Food Choice :: !!
2  Ticket Type [ Business ] >> Food Choice
```

Lastly, you can do a quick review of available options & syntax by clicking the "Usage" button in top-right section of the Advanced Mode:

The advanced constraints editor allows you to edit constraints as text and express more complex constraints.

The editor understands the contents of your test plan and provides intelligent auto-completion and constraint validation. The constraints being manipulated are all simply text, so you can use the normal copy and paste capabilities of your browser and operating system as needed.

A specific parameter value is referenced in a constraint with the text `Parameter [ Value ]`, and constraints can refer to more than one value of the same parameter with `Parameter [ Value1 , Value2 , ... , ValueN ]`.

The three types of basic constraints use this notation to constrain parameter values from two parameters. For example:

`Age [ Under 13 , Between 14 and 17 , Between 18 and 20 ] ≠ Beverage [ Alcohol ]`

The invalid constraint `≠` (can be typed as !=), the bound constraint `→` (can be typed as ->), and the mutually bound constraint `↔` (can be typed as <->).

An advanced skip constraint `>>` (can be typed as >>) is available only in this editor. The skip constraint allows you to specify a triggering condition to the left of the `>>` skip operator, and a parameter that should be skipped and should not achieve any coverage in test cases that satisfy the condition. For example:

`Hotel Reservation [ Not Added ] >> Hotel Quality`

With this constraint in place, in any test case where the parameter `Hotel Reservation` takes the value `Not Added`, then the parameter `Hotel Quality` will be skipped, meaning it will not receive any value or coverage. If a hotel is not added to the reservation, then a test is unable to select a hotel quality (e.g. 5-star). The parameter used in the condition must occur before any skipped parameters in the test plan.

You can also skip a range of parameters at once using the `::` range sytax. For example, to skip both the parameters `Hotel Quality` and `Hotel Price Range` and *all* the parameters that occur *between* them in the test plan, it looks like this:

`Hotel Reservation [ Do Not Add ] >> Hotel Quality :: Hotel Price Range`

An end of test case symbol `!!` can also be used in a skip constraint; it's a virtual parameter that exists after the very last parameter. For example, let's assume that everything following `Hotel Reservation` in the test plan is a hotel-related parameter. We can specify that if hotel is not added, then *all* of the parameters *after* `Hotel Reservation` should be skipped:

`Hotel Reservation [ Not Added ] >> !!`

The end of test case `!!` symbol can be used as the end of a range. For example, skipping both `Hotel Quality` and *all* of the paramters *after* it:

`Hotel Reservation [ Do Not Add ] >> Hotel Quality :: !!`

The skip constraint is the *only* aspect of a test plan that depends on the parameter order. The parameter referenced in the triggering condition (the left side of the `>>` operator) must occur before the parameters to be skipped (the ones on the right). Similarly, a range of parameters to be skipped must be specified with the earliest parameter first. For example, if a plan includes parameters A, B, and C, *in that order*, then it's not possible to specify a

▬ ▮ | Save (Cmd+S) | ❓ Usage

View a Constraint

① To view constraints, select the option "Rules" and then "Constraints" on the model

② A list with all the bound and invalid constraints will be displayed.

Edit a Constraint

To edit a constraint, you need to change view to "bulk" . Then you can edit the constrain type selecting the corresponding operator.

It is not possible to edit a constrain at the Standard View. In the Standard View you are only allowed to delete and create a new constrain.

Delete a Constraint

To delete a constraint, hover the constraint and select the delete option:

Another option is to, on bulk view mode, delete the desired constraint line: